ALGORITHMIQUE · CHAPITRE 28 · MISE EN PRATIQUE

Vérifier un résultat et traiter les cas limites

Définissez des résultats attendus, testez les frontières et utilisez une assertion. Distinguez erreur d'entrée, défaut du programme, test et preuve.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Établir un résultat attendu avant l'exécution.
- Couvrir vide, zéro, une seule valeur et plusieurs valeurs.
- Distinguer une entrée invalide d'un résultat incorrect.

Prérequis. Savoir parcourir une suite et lire une fonction avec résultat.

Un résultat plausible suffit-il ?

Une fonction compte les nombres strictement positifs d'une suite de réels finis. Pour `[-2, 0, 3, 3]`, le résultat attendu vaut 2 : zéro n'est pas positif et les deux occurrences de 3 sont comptées séparément. Un affichage "2" sur ce seul exemple ne suffit pas à vérifier toute la règle.

Choisir les cas avant le code

Un petit ensemble de vérifications justifiées

Entrée	Résultat attendu	Ce que le cas vérifie
[]	0	Aucune valeur à compter
[0]	0	Frontière : zéro est exclu

Entrée	Résultat attendu	Ce que le cas vérifie
[5]	1	Une seule valeur positive
[-2, 0, 3, 3]	2	Négatif, zéro et répétitions

Un résultat attendu se calcule indépendamment de la fonction testée. Si l'on copie la même logique erronée dans le calcul de référence, deux erreurs identiques peuvent sembler se confirmer. Commencer par des entrées petites et explicables à la main évite ce piège.

Définir une assertion avant de lancer les vérifications

Un **cas de test** associe une entrée précise à un résultat attendu. Une **assertion** est une vérification qui échoue si la condition annoncée est fausse. Python et MATLAB proposent `assert`. Par exemple, `assert count_positive([5]) == 1` demande que la fonction renvoie exactement un élément positif pour cette entrée.

La fonction `count_positive` compte les valeurs strictement supérieures à zéro dans une suite de nombres réels finis. Elle ne compte pas zéro. Pour [-2, 0, 3, 3], les deux occurrences de 3 comptent séparément : le résultat vaut 2. Une suite vide produit 0, car aucun élément n'a été compté.

Un **défaut** est un écart entre le programme et son contrat. Une entrée hors du contrat est une autre situation. Si l'on veut rejeter des textes, des valeurs non finies ou une forme de tableau particulière, il faut définir puis programmer cette validation. Le jeu d'essais montré ne transforme pas une fonction numérique simple en interface capable d'accepter toute donnée.

En Python, les assertions peuvent être désactivées par certaines options d'exécution. Elles servent ici à vérifier les exemples et ne doivent pas être l'unique contrôle d'une entrée externe. Chaque résultat attendu doit venir du problème ou d'un calcul indépendant ; copier le résultat du programme dans l'assertion ne vérifierait pas sa correction.

Vérifier le contrat de la fonction

Pseudo-code

```
// example.pseudo
// count_positive
FUNCTION count_positive(values)
    count <- 0
    FOR EACH value IN values DO
        IF value > 0 THEN
            count <- count + 1
        END IF
    END FOR
    RETURN count
END FUNCTION
```

Python

```python
# count_positive.py
def count_positive(values):
    count = 0
    for value in values:
        if value > 0:
            count = count + 1
    return count

assert count_positive([]) == 0
assert count_positive([0]) == 0
assert count_positive([5]) == 1
assert count_positive([-2, 0, 3, 3]) == 2
print("All cases passed.")
```

MATLAB

```
% count_positive.m
function count = count_positive(values)
    count = 0;
    for index = 1:numel(values)
        value = values(index);
        if value > 0
            count = count + 1;
        end
    end
end
```

`assert` vérifie ici une proposition. Si elle est fausse, la vérification signale un échec au lieu de continuer comme si le résultat était correct. Enregistrez la fonction MATLAB dans `count_positive.m` avant d'exécuter ses vérifications.

MATLAB

```
% check_count_positive.m
assert(count_positive([]) == 0);
assert(count_positive([0]) == 0);
assert(count_positive([5]) == 1);
assert(count_positive([-2, 0, 3, 3]) == 2);
disp('All cases passed.');
```

Faire échouer un cas utile

Remplacez strictement positif par `value ≥ 0` dans la condition. Le cas `[0]` révèle immédiatement la différence de contrat. Remettez ensuite le test correct, puis essayez une suite entièrement négative : le compteur doit rester zéro.

Définir le domaine et le comportement en dehors

Cette fonction suppose des nombres réels finis, pas des textes à convertir ni des valeurs manquantes. Si les données viennent d'une saisie ou d'un fichier, il faut définir leur format et les contrôler avant de les transmettre. Choisir de rejeter une entrée, de la corriger ou de l'ignorer relève du contrat ; les deux langages ne doivent pas silencieusement appliquer des règles différentes.

Expliquer ce que le test démontre

▼ **Pourquoi faut-il garder les deux occurrences de 3 dans le résultat ?**

Le problème compte les éléments positifs, pas les valeurs positives distinctes. Remplacer la suite par un ensemble de valeurs uniques modifierait le problème. Le cas avec répétitions permet de rendre cette distinction visible.

▼ **Des tests réussis prouvent-ils que tous les résultats sont corrects ?**

Non. Ils montrent que les cas essayés respectent les attentes choisies. Un raisonnement sur la boucle explique plus généralement pourquoi le compteur représente le nombre d'éléments positifs déjà parcourus.

Associer chaque cas à une question

Un bon cas de vérification a une raison d'être : une frontière, une branche, un vide, une répétition ou une entrée interdite. Écrivez ce qu'il vérifie pour éviter d'accumuler des essais sans savoir quelle erreur ils pourraient révéler.

À retenir - fiche mémo

Test

Une entrée et une attente définies avant l'exécution.

Assertion

Une vérification qui signale l'échec d'une condition attendue.

Limites

Vide, zéro et frontières répondent à des questions différentes.

Pseudo-code

```
// example.pseudo
// count_positive_expectations
// Input []: expected 0.
// Input [0]: expected 0.
// Input [-2, 0, 3, 3]: expected 2.
```

Point de vigilance. Les assertions Python peuvent être désactivées ; elles ne remplacent pas une validation d'entrée externe.

Consulter la cheatsheet : Vérifications et projets · **Télécharger la fiche du thème en PDF**

Version web : https://www.clearcode.fr/courses/programming/algorithmic/practice/verification