ALGORITHMIQUE · CHAPITRE 21 · FONCTIONS

Portée et état des variables

Comprenez variables locales, alias et modification de données. Comparez le partage d'une liste Python et la valeur d'un vecteur numérique MATLAB.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Identifier les variables locales à un appel.
- Comprendre le partage d'une liste Python.
- Comparer ce partage avec la sémantique de valeur d'un vecteur MATLAB.

Prérequis. Comprendre paramètres, arguments et valeur de retour.

Le résultat est-il la seule chose qui change ?

Une fonction remplace la première valeur d'une suite non vide par 9. Elle renvoie une suite mise à jour. Mais la variable d'origine est-elle restée intacte ? Pour répondre, connaître le nom de la fonction ne suffit pas : il faut examiner le contrat et les règles du langage.

Un nom local n’est pas une protection contre tous les effets

Le paramètre `values` est un nom local à l'appel. En Python, il peut néanmoins désigner la même liste mutable que la variable de l'appelant. Modifier un élément de cette liste est alors visible à travers les deux noms. Réaffecter seulement le nom local à une autre liste serait une opération différente.

Définir portée, alias et modification en place

La **portée** d'un nom est la partie du programme où ce nom peut être utilisé. Un nom **local** à une fonction appartient à l'exécution de cette fonction. Deux appels peuvent donc employer le même nom `values` tout en recevant des données différentes. Cette règle sur les noms ne suffit pas à décrire le partage éventuel des données.

Un **objet mutable** est une donnée dont le contenu peut être modifié après sa création. Une liste Python en est un exemple. Une **référence** permet à un nom de désigner un objet. Deux noms qui désignent le même objet sont des **alias**. Après `alias = original`, modifier un élément de la liste Python est observable par les deux noms.

Une **modification en place** change l'objet existant, par exemple son premier élément. Une **réaffectation** associe un nom à une autre valeur. Ces actions sont différentes : remplacer localement le nom par une nouvelle liste ne modifie pas le contenu de l'ancienne liste. Le contrat doit donc dire ce qui change et pour qui.

Le vecteur numérique MATLAB de ce chapitre se comporte comme une valeur : la fonction retourne un vecteur modifié, tandis que les variables de l'appelant conservent leurs valeurs initiales. Cette observation porte sur les tableaux numériques montrés ici ; elle ne décrit pas tous les objets MATLAB. `replace_first` exige au moins un élément, car lire ou remplacer la première position d'une suite vide ne fait pas partie de son contrat.

Comparer les effets annoncés

Pseudo-code

```
// example.pseudo
// replace_first
FUNCTION replace_first(values, replacement)
    values[0] <- replacement
    RETURN values
END FUNCTION
// The effect on the input depends on the data-passing contract.
```

Python

```python
# replace_first.py
def replace_first(values, replacement):
    values[0] = replacement
    return values

original = [1, 2]
alias = original
updated = replace_first(original, 9)
print(original, alias, updated)
```

MATLAB

```matlab
% replace_first.m
function values = replace_first(values, replacement)
    values(1) = replacement;
end
```

Les fonctions prennent ici des suites non vides. Le pseudo-code indique le changement d'un élément et le résultat renvoyé ; les deux implémentations précisent comment ce changement est observé. Enregistrez la définition MATLAB dans `replace_first.m`, puis lancez cet exemple.

MATLAB

```matlab
% use_replace_first.m
original = [1, 2];
alias = original;
updated = replace_first(original, 9);
disp(original); % [1 2]
disp(alias);    % [1 2]
disp(updated);  % [9 2]
```

Lire les valeurs après l’appel

Le résultat correspond, mais les effets sur l'entrée diffèrent

Langage	original	alias	updated
Python	[9, 2]	[9, 2]	[9, 2]
MATLAB	[1, 2]	[1, 2]	[9, 2]

Dans la version Python, original et alias désignent la même liste ; updated est aussi cette liste renvoyée. Dans la version MATLAB, les vecteurs numériques ordinaires ont une sémantique de valeur : la modification locale ne change pas les valeurs conservées chez l'appelant. Les optimisations de copie mémoire ne modifient pas ce contrat observable.

Choisir le comportement recherché

Pour préserver la liste originale en Python, l'appelant peut transmettre `original.copy()` à cette fonction. Il obtient alors une nouvelle liste dont les modifications ne touchent pas l'original. Cette copie est superficielle ; notre exemple ne contient que des nombres, pas des objets imbriqués. En MATLAB, réaffecter `original = replace_first(original, 9)` choisit explicitement de remplacer la valeur de la variable.

Distinguer modification et réaffectation

▼ Un même nom values dans deux appels signifie-t-il que les appels partagent forcément leurs données ?

Non. Chaque appel possède ses paramètres locaux. Les données peuvent être partagées ou non selon ce qui est transmis et les règles du langage. Il faut suivre les valeurs et les objets concernés, pas seulement l'orthographe des noms.

▼ **Pourquoi une suite vide est-elle hors du contrat de cet exemple ?**

La méthode accède directement au premier élément. Pour accepter le vide, il faudrait choisir un comportement, puis le traiter avant cet accès : par exemple signaler une erreur ou renvoyer la suite vide. Le choix ne doit pas rester implicite.

Décrire aussi les effets

Le contrat d'une fonction peut inclure ses entrées, son résultat et les modifications qu'elle produit. Un nom local et une valeur retournée ne suffisent pas à promettre l'absence d'effets sur une donnée partagée. Cette distinction aide à lire ensuite les états séparés des appels récursifs.

À retenir - fiche mémo

Portée

La région du programme dans laquelle un nom peut être utilisé.

Alias Python

Deux noms peuvent désigner la même liste mutable ; une modification devient visible par les deux.

Vecteur MATLAB

Dans cet exemple numérique, l’appelant conserve l’entrée et reçoit une valeur modifiée en sortie.

Pseudo-code

```
// example.pseudo
// replace_first
// Requires at least one element.
FUNCTION replace_first(values, replacement)
    values[0] <- replacement
    RETURN values
END FUNCTION
```

Point de vigilance. Un nom local n’empêche pas à lui seul une modification d’un objet partagé.

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

Version web : https://www.clearcode.fr/courses/programming/algorithmic/functions/scope-and-state