ALGORITHMIQUE · CHAPITRE 20 · FONCTIONS

Paramètres, arguments et résultat

Distinguez paramètre, argument, affichage et valeur de retour. Suivez les données d'un calcul de commande dans une fonction Python ou MATLAB.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Distinguer un paramètre d'un argument.
- Suivre l'ordre des valeurs transmises.
- Réutiliser un résultat sans l'afficher dans la fonction.

Prérequis. Avoir défini et appelé une première fonction.

Une même règle, plusieurs commandes

Un total dépend d'un prix unitaire, d'une quantité et de frais fixes. Écrire une fonction qui utilise directement 250, 3 et 100 ne la rendrait utile que pour un exemple. Nous voulons transmettre ces données à chaque appel, en conservant le calcul `price × quantity + delivery`. Les montants de l'exemple sont en centimes entiers.

Paramètre et argument

Un **paramètre** est un nom dans la définition : price, quantity ou delivery. Un **argument** est la valeur transmise lors d'un appel. Dans `compute_total(250, 3, 100)`, le paramètre price reçoit 250, quantity reçoit 3 et delivery reçoit 100. L'ordre est donc une partie de l'interface.

Définir les rôles des trois entrées

La fonction `compute_total` reçoit un prix unitaire `price` en centimes, une quantité entière non négative `quantity` et des frais `delivery` en centimes. Elle retourne le prix multiplié par la quantité, augmenté des frais. Le prix et les frais sont non négatifs dans les exemples. La fonction ne décide pas elle-même d'une remise ou d'une livraison gratuite.

Les arguments sont **positionnels** : leur place détermine le paramètre qui les reçoit. Pour `compute_total(250, 3, 100)`, 250 devient le prix, 3 la quantité et 100 les frais. Échanger les deux derniers nombres change le problème calculé, même si le programme reste syntaxiquement exécutable.

Un **effet observable** est un changement que l'on peut constater en dehors du calcul local, comme un affichage. Afficher 850 montre un texte à l'écran ; retourner 850 transmet un nombre réutilisable dans un autre calcul. Une fonction peut afficher sans renvoyer le résultat souhaité. En Python, une fonction qui se termine sans `return` explicite renvoie `None`, une valeur spéciale qui représente ici l'absence de résultat numérique.

Pour éviter une ambiguïté commerciale, l'exemple conserve les frais même lorsque la quantité vaut zéro : `compute_total(250, 0, 100)` vaut 100. Le mini-projet final choisira une autre règle pour une commande vide. Les deux règles ne se contredisent pas si leurs contrats sont annoncés avant le code.

Renvoyer le calcul

Pseudo-code

```
// example.pseudo
// compute_total
FUNCTION compute_total(price, quantity, delivery)
    RETURN price * quantity + delivery
END FUNCTION
```

Python

```python
# compute_total.py
def compute_total(price, quantity, delivery):
    return price * quantity + delivery

total = compute_total(250, 3, 100)
print(total)  # 850
```

MATLAB

```matlab
% compute_total.m
function total = compute_total(price, quantity, delivery)
    total = price * quantity + delivery;
end
```

La fonction ne contient aucun affichage. Le résultat 850 est d'abord transmis à l'appelant, qui le conserve dans `total`. En MATLAB, gardez la définition dans `compute_total.m` ; les appels se font dans un autre script ou dans la fenêtre de commande.

MATLAB

```matlab
% use_compute_total.m
total = compute_total(250, 3, 100);
disp(total); % 850
```

Suivre deux appels distincts

Le même corps, des arguments différents

Appel	Calcul	Résultat
compute_total(250, 3, 100)	250 × 3 + 100	850
compute_total(250, 1, 100)	250 × 1 + 100	350

Appel	Calcul	Résultat
compute_total(250, 0, 100)	250 × 0 + 100	100

Le troisième appel conserve les frais fixes lorsque la quantité est zéro. C'est bien la règle codée, même si une autre règle commerciale pourrait prévoir une commande vide sans frais. Le contrat permet de décider si ce comportement est attendu.

Afficher n’est pas retourner

Afficher produit une information visible. Retourner fournit une valeur à la suite du calcul. Si une fonction Python se contente d'un `print` et ne retourne rien explicitement, sa valeur de retour est `None`. En MATLAB, demander une sortie à une fonction déclarée sans sortie ne transforme pas son affichage en résultat : il faut déclarer et affecter la sortie prévue.

Vérifier les dépendances

▼ **Peut-on écrire compute_total(250, 3, 100) + 50 ?**

Oui : la fonction renvoie 850, puis l'expression ajoute 50 et produit 900. L'affichage éventuel peut rester à l'extérieur. C'est précisément ce que permet une valeur de retour.

▼ **Pourquoi le contrat doit-il préciser l'ordre des arguments ?**

Intervertir quantity et delivery changerait le calcul. Même si certains jeux de nombres peuvent produire le même résultat par hasard, une interface doit associer chaque valeur à son rôle sans dépendre de cette coïncidence.

Un appel est un contrat local

Avant d'appeler une fonction, identifiez les valeurs disponibles. Après l'appel, dites quelle valeur a été produite et où elle est conservée. La portée des noms et les effets sur les données demandent une distinction supplémentaire, abordée au chapitre suivant.

À retenir - fiche mémo

Ordre

Un argument positionnel est associé au paramètre occupant la même position.

Résultat

Le retour peut être affecté à une variable ou utilisé dans une expression.

Règle

compute_total ajoute les frais annoncés, même avec une quantité nulle.

Pseudo-code

```
// example.pseudo
// compute_total
FUNCTION compute_total(price, quantity, delivery)
    RETURN price * quantity + delivery
END FUNCTION
```

Point de vigilance. Ne confondez pas cette règle avec celle du mini-projet, où la commande vide coûte zéro.

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

Version web : https://www.clearcode.fr/courses/programming/algorithmic/functions/parameters-and-return