ALGORITHMIQUE · FICHE PRATIQUE

Cheatsheet : Les essentiels de l'algorithmique

Distinguer problème, algorithme et implémentation, puis vérifier un contrat.

Par Ali ZAINOUL · Cours gratuit

Un algorithme est une méthode composée d'étapes précises pour obtenir un résultat à partir de données. Une recette vague telle que "calculer la durée" ne suffit pas : il faut savoir ce qui entre, ce qui sort et quelles règles s'appliquent.

Cette fiche permet de construire et vérifier un premier calcul, même sans langage de programmation connu. Les explications sont en français ; les noms et les instructions des blocs sont en anglais.

1. Définir le problème avant la méthode

Une entrée est une donnée fournie au traitement. Une sortie est un résultat produit. Une contrainte limite les données autorisées ou précise une règle. L'ensemble constitue le contrat du calcul.

Exemple : recevoir un nombre entier non négatif de secondes et produire des minutes complètes ainsi qu'un reste inférieur à 60 secondes. Les nombres négatifs ne font pas partie de cet exemple. Une durée de 125 secondes doit donner 2 minutes et 5 secondes, car 2 × 60 + 5 = 125.

Le contrat de la conversion

Question	Réponse précise
Entrée	total_seconds : entier non négatif, en secondes
Sorties	minutes : minutes complètes ; seconds : reste en secondes
Propriété à conserver	minutes × 60 + seconds = total_seconds

Question	Réponse précise
Frontière du reste	0 ≤ seconds < 60
Exemple attendu	125 → 2 minutes, 5 secondes

2. Lire une instruction sans deviner les mots

Une variable associe un nom à une valeur. Une affectation enregistre une valeur sous ce nom ; ce n'est pas une égalité à résoudre. Une expression calcule une valeur à partir d'autres valeurs. Les parenthèses d'un appel regroupent les données transmises à une opération nommée.

Le pseudo-code est une notation explicative, pas un programme directement exécutable. Dans cette fiche, floor_divide donne le quotient entier arrondi vers le bas, MOD donne le reste et PRINT affiche. Les scripts Python et MATLAB fixent leur entrée dans le fichier : aucune saisie n'est nécessaire.

Lire les premières lignes

Écriture	Signification
total_seconds <- 125	Associer le nombre 125 au nom total_seconds
CONSTANT SECONDS_PER_MINUTE <- 60	Annoncer une valeur que la méthode ne modifiera pas
floor_divide(125, 60)	Compter les groupes complets de 60 : résultat 2
125 MOD 60	Calculer le reste après ces groupes : résultat 5
// commentaire	Texte explicatif ignoré par le calcul

3. Un exemple complet dans trois écritures

Une constante est une valeur que l'on décide de conserver pendant le traitement. Ici, une minute contient toujours 60 secondes. Les majuscules du nom sont une convention ; elles ne bloquent pas techniquement une réaffectation dans les scripts Python et MATLAB.

En Python, // effectue la division entière et % le modulo. En MATLAB, floor arrondit vers le bas et mod calcule le reste. Les petits entiers choisis sont exactement représentables dans ces exemples. Un commentaire commence par # en Python et par % en MATLAB.

Trace des deux calculs

Après l’étape	minutes	seconds
Entrée fixée à 125	Non calculé	Non calculé
Quotient entier	2	Non calculé
Reste	2	5

Convertir 125 secondes

Enregistrez le bloc du langage choisi sous le nom indiqué dans son premier commentaire, puis exécutez ce script.

Pseudo-code

```
// example.pseudo
// convert_duration
CONSTANT SECONDS_PER_MINUTE <- 60
total_seconds <- 125
minutes <- floor_divide(total_seconds, SECONDS_PER_MINUTE)
seconds <- total_seconds MOD SECONDS_PER_MINUTE
PRINT minutes, seconds
```

`CONSTANT` annonce la règle ; `PRINT` rend les deux résultats visibles.

Python

```
# convert_duration.py
SECONDS_PER_MINUTE = 60
total_seconds = 125
minutes = total_seconds // SECONDS_PER_MINUTE
seconds = total_seconds % SECONDS_PER_MINUTE
print(minutes, seconds)
```

`print` affiche 2 puis 5 sur une ligne ; ce sont deux résultats numériques calculés avant l'affichage.

MATLAB

```
% convert_duration.m
SECONDS_PER_MINUTE = 60;
total_seconds = 125;
minutes = floor(total_seconds / SECONDS_PER_MINUTE);
seconds = mod(total_seconds, SECONDS_PER_MINUTE);
disp([minutes, seconds]);
```

`floor` calcule les minutes complètes ; `mod` donne le reste ; `disp` affiche le vecteur contenant 2 et 5.

> **Résultat expliqué.** Le résultat est 2 minutes et 5 secondes. Le total initial reste égal à 125 ; aucune unité n’a été perdue.

4. Vérifier le calcul et ses limites

Un cas de test associe une entrée précise à un résultat attendu, calculé indépendamment du programme. Essayez une entrée ordinaire, puis les frontières où le comportement change. Une sortie qui reconstitue la durée n'est pas forcément conforme : 1 minute et 65 secondes fait aussi 125 secondes, mais viole la règle du reste.

Un test réussi ne démontre pas tous les cas possibles. Ici, le raisonnement s'appuie sur la décomposition d'une durée en groupes de 60 et un reste : les deux propriétés du contrat doivent être vérifiées ensemble.

Attentes à calculer avant exécution

Secondes reçues	Minutes complètes	Reste
0	0	0
59	0	59
60	1	0
61	1	1
125	2	5

> **Erreur à éviter.** Diviser par 60 puis conserver la partie décimale ne donne pas directement un nombre de secondes restantes.

▼ Que doit produire une entrée de 3 601 secondes ? Pourquoi 60 minutes et 61 secondes est-il incorrect ?

60 minutes et 1 seconde : 60 × 60 + 1 = 3 601 et le reste appartient à [0, 60[. Le second résultat représente 3 661 secondes et son reste dépasse la frontière autorisée.

Télécharger cette fiche en PDF · **Toutes les fiches par thème**

Version web : https://www.clearcode.fr/courses/programming/algorithmic/cheatsheet/essentials