ALGORITHMIQUE · CHAPITRE 1 · FONDATIONS

Passer d'un énoncé à un problème précis

Apprenez à définir les entrées, les sorties et les contraintes d'un algorithme avec une conversion de secondes en minutes, expliquée et vérifiée.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Distinguer une demande et un problème suffisamment précis.
- Identifier entrées, sortie et conventions.
- Décomposer un calcul en étapes vérifiables.

Prérequis. Aucun prérequis de programmation.

Que signifie "convertir une durée" ?

Une durée de 125 secondes correspond à 2 minutes et 5 secondes. Pour obtenir ce résultat avec une méthode reproductible, que faut-il recevoir, calculer et afficher ? Le problème semble simple, mais "convertir une durée" ne précise ni l'unité d'entrée, ni l'unité de sortie, ni le traitement des valeurs négatives.

Rendre la demande précise

Une **entrée** est une donnée reçue par la méthode ; une **sortie** est le résultat recherché. Nous choisissons ici un nombre entier de secondes, supérieur ou égal à zéro. La sortie contient deux entiers : les minutes complètes et les secondes restantes. Cette convention ne décrit pas encore le calcul ; elle précise ce que le résultat doit représenter.

Le contrat de la conversion

Élément	Convention choisie
Entrée	125 secondes, entier non négatif
Sortie	2 minutes et 5 secondes
Limite du reste	De 0 à 59 secondes
Vérification	2 × 60 + 5 = 125

Découper sans perdre l’unité

Une minute contient 60 secondes. Nous cherchons d'abord combien de groupes complets de 60 tiennent dans la durée, puis ce qui reste. Le **quotient entier** compte les minutes complètes ; le **reste** compte les secondes encore disponibles. Le quotient n'est pas une durée décimale : 125 / 60 n'est pas le nombre de minutes complètes.

Lire un premier exemple sans prérequis

Un **nombre entier** est un nombre sans partie fractionnaire, comme 0, 2 ou 125. Ici, chaque nombre est accompagné d'une unité : des secondes ou des minutes. Le calcul conserve la même durée ; seule sa représentation change. **Non négatif** signifie supérieur ou égal à zéro. Ce choix exclut les durées négatives de notre exemple.

Une **variable** donne un nom à une valeur utilisée dans le calcul. `total_seconds` désigne la durée de départ ; `minutes` et `seconds` désignent les deux résultats. Une **affectation** enregistre une valeur sous ce nom : `total_seconds = 125` fixe notre donnée de démonstration. Le chapitre suivant expliquera comment cette valeur peut changer.

Le **pseudo-code** décrit les étapes avec une écriture choisie pour le cours, sans être un programme directement exécutable. Nous utilisons l'anglais dans les blocs : `READ` signifie recevoir une entrée ; `PRINT` signifie afficher. La flèche `<-` affecte le résultat de droite au nom de gauche. Dans les versions exécutables, l'entrée est déjà fixée à 125 : aucune saisie au clavier n'est requise.

`floor_divide(125, 60)` désigne le quotient entier, ici 2. `125 MOD 60` désigne le reste, ici 5. En Python, les écritures correspondantes sont `//` et `%`. En MATLAB, `floor` arrondit le quotient vers le bas et `mod` calcule le reste. Un nom suivi de parenthèses, comme `floor(125 / 60)`, demande une opération sur les valeurs placées entre ces parenthèses : c'est un **appel de fonction**. Nous apprendrons ensuite à écrire nos propres fonctions.

Un **commentaire** explique le code sans participer au calcul : il commence par `//` dans notre pseudo-code, `#` en Python et `%` en MATLAB. Le premier commentaire indique le nom de l'exemple ou du fichier. Un **script** est un fichier contenant des instructions exécutées dans l'ordre. Enregistrez la version choisie dans `convert_duration.py` ou `convert_duration.m`. Le point-virgule MATLAB supprime l'affichage automatique d'une ligne ; `disp` affiche volontairement le résultat.

Écrire les étapes

Pseudo-code

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

Python

```python
# convert_duration.py
total_seconds = 125
minutes = total_seconds // 60
seconds = total_seconds % 60
print(minutes, seconds)  # 2 5
```

MATLAB

```
% convert_duration.m
total_seconds = 125;
minutes = floor(total_seconds / 60);
seconds = mod(total_seconds, 60);
disp([minutes, seconds]); % [2 5]
```

La première ligne commentée nomme l'exemple ; elle n'est pas une instruction de calcul. Les versions utilisent les mêmes noms. `//` en Python et `floor` en MATLAB réalisent ici le quotient entier ; leur signification sera reprise avec les expressions. `PRINT`, `print` et `disp` rendent un résultat visible.

Prévoir puis contrôler

Avant de modifier 125, prévoyez les résultats pour 0, 59, 60 et 61. Exécutez ensuite le calcul, à la main ou avec le langage choisi. Vérifiez deux propriétés : `minutes × 60 + seconds = total_seconds` et `0 ≤ seconds < 60`. Une sortie peut satisfaire l'égalité tout en violant la seconde propriété : 1 minute et 65 secondes représente aussi 125 secondes, mais ne respecte pas notre contrat.

Préciser un autre énoncé

▼ **Que faut-il préciser pour "calculer un prix total" ?**

Il faut notamment connaître les quantités, les prix et leurs unités, puis préciser si des frais ou remises s'appliquent. Une fois les conventions choisies, on peut calculer un exemple de référence à la main. Il n'existe pas une réponse unique tant que la demande reste ambiguë.

▼ **Quel résultat attend-on pour 3600 secondes ?**

60 minutes et 0 seconde. Notre sortie ne comporte pas de champ heures : répondre 1 heure modifierait le contrat. C'est un autre problème, même si la durée représentée est équivalente.

Ce que l'on peut déjà expliquer

Vous savez décrire les données et le résultat attendu avant de coder. Une bonne question de départ est : "Comment pourrai-je reconnaître une réponse correcte ?" Le chapitre suivant explique comment une variable conserve une valeur pendant les étapes du calcul.

À retenir - fiche mémo

Entrée

Une durée entière non négative, exprimée en secondes.

Sortie

Des minutes complètes et un reste de 0 à 59 secondes.

Contrôle

minutes * 60 + seconds doit reconstituer la durée reçue.

Pseudo-code

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

Point de vigilance. 1 minute et 65 secondes représente 125 secondes mais ne respecte pas le contrat du reste.

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

Version web : https://www.clearcode.fr/courses/programming/algorithmic/foundations/problem-solving