EXPLIQUÉ VISUELLEMENT

# Les algorithmes en clair

Un voyage visuel des structures de données à l'IA générative



CHORAL SOFTWARE

# Droits d'auteur

Copyright © 2026 Choral Software. Tous droits réservés. Aucune partie de cette publication ne peut être reproduite, stockée, ou transmise sous quelque forme ou par quelque moyen que ce soit sans autorisation écrite préalable, sauf pour de brèves citations utilisées dans des critiques ou des travaux universitaires.

## Avis pédagogique

Ce livre explique les algorithmes et l'intelligence artificielle à des fins d'éducation générale. Il ne fournit pas de conseils médicaux, juridiques, financiers, de sécurité, ou autres conseils professionnels. Les exemples sont simplifiés pour faciliter la compréhension ; les systèmes en production exigent des tests, une expertise du domaine, des garanties de confidentialité, une revue de sécurité, et une surveillance continue.

Édition 1.1 • Juillet 2026

## Note sur cette édition

Cette édition corrige le pseudocode illustratif dans l'ensemble du livre. Dans la première impression, la plupart des blocs de pseudocode étaient un texte générique plutôt que la logique réelle de l'algorithme nommé. Chaque bloc de pseudocode a été réécrit avec une logique correcte, proche d'un vrai code, pour son concept (ou, pour un petit nombre de sections purement conceptuelles, avec un exemple concret illustratif). Quelques consignes de réflexion de la partie V qui évoquaient la « mémoire » et la « croissance » d'une manière inadaptée à des sujets non algorithmiques ont également été reformulées pour convenir à leur sujet.

# Comment lire ce livre

Comment lire ce livre. Vous n'avez pas besoin de connaître la programmation ou les mathématiques avancées. Commencez par les images et les analogies. Lisez l'explication en langage clair. Puis suivez le pseudocode comme s'il s'agissait d'indications de mise en scène. Le pseudocode est intentionnellement proche de Python, mais il est conçu pour être lu plutôt qu'exécuté. Chaque concept occupe une double page d'apprentissage. La première page construit le modèle mental ; la seconde transforme l'idée en laboratoire visuel. La couleur est fonctionnelle : le bleu marque la structure, le turquoise marque la transformation, l'or marque le choix ou la priorité, et le corail marque le risque ou l'erreur. Une règle de lecture utile : ne demandez pas « Est-ce que je comprends chaque ligne ? » Demandez « Puis-j'expliquer quelle information la méthode conserve, quelle action répétée elle effectue, et pourquoi elle s'arrête ? » Une note sur l'IA : la partie consacrée à l'IA est délibérément étendue. Elle commence par les données et l'évaluation, traverse l'apprentissage automatique classique et les réseaux de neurones, puis développe la vision, le langage, les transformeurs, l'IA générative, la récupération, les recommandations, la parole, l'apprentissage par renforcement, les agents, et le déploiement responsable. Les systèmes du quotidien sont traités comme des chaînes de traitement plutôt que comme des modèles uniques mystérieux.

# Table des matières


## PARTIE I: Penser en structures

1. Les algorithmes sans intimidation
2. Données : valeurs, enregistrements et références
3. Tableaux et listes dynamiques
4. Listes chaînées
5. Piles, files et deques
6. Tables d'hachage, ensembles et dictionnaires

## PARTIE II: Arbres, graphes, et la forme des relations

7. Arbres et hiérarchies
8. Arbres binaires de recherche et équilibre
9. Tas et files de priorité
10. Graphes : des réseaux partout
11. Parcourir les graphes

## PARTIE III: Algorithmes à usage général

12. La recherche
13. Le tri élémentaire
14. Le tri rapide
15. Diviser, combiner, et réutiliser
16. Algorithmes gloutons et planification
17. Plus courts chemins et arbres couvrants

## PARTIE IV: Comment les machines apprennent

18. IA, apprentissage automatique, et modèles
19. Préparation des données et évaluation
20. Régression : prédire des quantités
21. Classification : choisir une catégorie
22. Arbres de décision et ensembles
23. Voisins, groupes, et similarité
24. Probabilité, mise à jour bayésienne, et incertitude
25. Les réseaux de neurones de l'intérieur
26. Entraîner des réseaux profonds
27. La vision par ordinateur
28. Le langage comme donnée
29. L'attention et les transformeurs
30. Les grands modèles de langage et l'IA générative
31. Récupération, recherche, et RAG
32. Systèmes de recommandation
33. IA de la parole et de l'audio
34. L'apprentissage par renforcement
35. Agents d'IA, outils, et planification

## PARTIE V: L'IA dans la vie quotidienne — et comment y réfléchir clairement

36. Cartes, navigation, et livraison
37. Fils d'actualité, publicités, et achats en ligne
38. Pourriel, fraude, modération, et sécurité
39. Équité, confidentialité, et IA responsable
40. Utiliser l'IA avec sagesse

PARTIE I

# Penser en structures

*Avant qu'un algorithme puisse agir, l'information a besoin d'une forme.*

CHAPITRE 1

# Les algorithmes sans intimidation

## Dans ce chapitre

**1.1 Une recette pour décider.** Un algorithme est une séquence finie et non ambiguë de choix et d'actions.

**1.2 Entrées, sorties et état.** Les entrées sont ce que la procédure reçoit ; les sorties sont ce qu'elle produit ; l'état est ce qui change en cours de route.

**1.3 La justesse avant l'ingéniosité.** Une procédure utile doit résoudre le problème promis pour toute entrée autorisée, pas seulement pour un exemple favorable.

**1.4 L'efficacité comme récit de croissance.** L'efficacité demande comment le travail croît quand l'entrée grandit, sans exiger de mathématiques avancées.

## Boussole du chapitre

En lisant Les algorithmes sans intimidation, comparez les quatre idées de ce chapitre : Une recette pour décider, Entrées, sorties et état, La justesse avant l'ingéniosité, L'efficacité comme récit de croissance. Traitez chaque bloc de pseudocode comme une carte de décisions, pas comme un test de programmation.

### 1.1 Une recette pour décider

# Une recette pour décider

*Un algorithme est une séquence finie et non ambiguë de choix et d'actions.*



*Figure 1.1. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Un point de départ intuitif. Commencez par suivre une recette de petit-déjeuner. Dans cette scène, remarquez ce qui peut bouger, ce qui sert de limite, et ce qui compte comme réussite. Ces mêmes questions révèlent l'idée centrale d'une recette pour décider : Un algorithme est une séquence finie et non ambiguë de choix et d'actions.

Le point de vue de la machine. Un ordinateur ne reçoit ni la métaphore ni son bon sens. Il reçoit une description structurée choisie pour cette tâche. Dans une recette pour décider, cette description doit soutenir l'opération impliquée par la définition ; dans transformer un objectif flou en petites étapes vérifiables, elle doit aussi refléter les unités, les valeurs manquantes et le sens de chaque lien.

L'invariant à surveiller. Pendant une recette pour décider, un fait doit rester fiable même quand le reste de l'information change. Ce fait peut être une région ordonnée, un ensemble de sommets visités, un meilleur score connu, ou une frontière autour du travail non résolu. Le nommer rend à la fois la procédure et sa condition d'arrêt plus faciles à vérifier.

## SOUS LE CAPOT | Une recette pour décider

À l'intérieur d'un système réel. Une recette pour décider contribue à transformer un objectif flou en petites étapes vérifiables, mais ce n'est qu'une étape. Un usage déployé d'une recette pour décider rassemble aussi et vérifie les entrées, applique des règles, gère les échecs, présente les résultats, et observe ce qui se passe ensuite. Les étapes environnantes déterminent souvent si un algorithme correct devient une expérience utile.

Ce qui mérite la suspicion. Un exemple bien propre peut cacher un coût, une ambiguïté, ou une hypothèse fragile. Pour une recette pour décider, examinez l'entrée la plus défavorable et toute égalité ou condition limite. Dans transformer un objectif flou en petites étapes vérifiables, demandez-vous aussi quelles erreurs sont les plus conséquentes et quel chemin de récupération existe.

### Traduction quotidienne

*Pensez à suivre une recette de petit-déjeuner. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
algorithm make_tea(water, tea_bag):
heat water until boiling
place tea_bag in cup
pour in water
wait 3 minutes
remove tea_bag
return cup
```

## LABORATOIRE VISUEL | Une recette pour décider

### 1. Traduire l'analogie

Esquissez suivre une recette de petit-déjeuner. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative d'une recette pour décider. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Imaginez l'exemple de suivre une recette de petit-déjeuner grandir jusqu'à ne plus tenir en mémoire. Décidez ce qui pourrait être traité par morceaux.

### 4. Remettre en question une hypothèse

Créez un contre-exemple qui rend un usage naïf d'une recette pour décider lent, ambigu, ou trompeur.

### 5. Localiser les choix humains

Dans un système impliquant transformer un objectif flou en petites étapes vérifiables, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Une recette pour décider ainsi : Un algorithme est une séquence finie et non ambiguë de choix et d'actions.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait une recette pour décider ?
2. Quel détail de « suivre une recette de petit-déjeuner » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

1.2 Entrées, sorties et état

# Entrées, sorties et état

*Les entrées sont ce que la procédure reçoit ; les sorties sont ce qu'elle produit ; l'état est ce qui change en cours de route.*

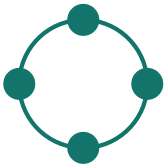


*Figure 1.2. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Voir la règle avant la notation. L'image d'un distributeur automatique qui se souvient de votre crédit donne à entrées, sorties et état une forme concrète. Les entrées sont ce que la procédure reçoit ; les sorties sont ce qu'elle produit ; l'état est ce qui change en cours de route. Nous garderons la structure utile de la comparaison tout en écartant les détails qui n'appartiennent pas à l'algorithme.

Choisir ce qu'il faut mémoriser. Une implémentation d'entrées, sorties et état repose sur un modèle délibérément choisi de la situation. Le meilleur modèle n'est pas le plus détaillé ; c'est la représentation la plus simple qui conserve les décisions que la méthode doit prendre. C'est particulièrement important pour séparer ce qui entre, ce qui est mémorisé, et ce qui sort, où un détail superflu peut ajouter du coût ou du bruit.

Suivre un passage complet. Partez d'un état initial valide, appliquez l'action définissant entrées, sorties et état, et notez ce que cette action établit. L'étape suivante doit exploiter cette nouvelle connaissance plutôt que de repartir de zéro. Le progrès devient visible quand la partie non résolue rétrécit, quand la frontière avance, ou quand l'estimation s'améliore.

## SOUS LE CAPOT | Entrées, sorties et état

Pourquoi cette idée se transpose bien. La même promesse structurelle rend entrées, sorties et état utile pour séparer ce qui entre, ce qui est mémorisé, et ce qui sort. Les noms des objets peuvent changer, mais l'opération reste reconnaissable. Ce transfert — d'une histoire de surface à une autre — est le signe que l'on a appris l'algorithme plutôt que mémorisé un exemple.

Des compromis, pas de la magie. Entrées, sorties et état gagne quelque chose — vitesse, structure, adaptabilité, ou une estimation utile — en s'appuyant sur des hypothèses particulières. Quand ces hypothèses échouent, le résultat peut devenir lent ou trompeur pour séparer ce qui entre, ce qui est mémorisé, et ce qui sort. Une implémentation robuste détecte les cas non pris en charge plutôt que d'inventer silencieusement une confiance.

### Traduction quotidienne

*Pensez à un distributeur automatique qui se souvient de votre crédit. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
function running_total(readings):
    total = 0
    for value in readings:
        total = total + value
    return total
# input: readings (what the procedure receives)
# state: total (what changes along the way)
# output: the final total (what it produces)
```

## LABORATOIRE VISUEL | Entrées, sorties et état

### 1. Traduire l'analogie

Esquissez un distributeur automatique qui se souvient de votre crédit. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative d'entrées, sorties et état. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Comparez un cas d'entrées, sorties et état presque résolu à un cas délibérément difficile. Expliquez pourquoi leur comportement d'exécution peut différer.

### 4. Remettre en question une hypothèse

Testez entrées, sorties et état avec un cas vide, un cas unique, des égalités, et une valeur malformée. Écrivez la réponse la plus sûre pour chacun.

### 5. Localiser les choix humains

Dans un système impliquant séparer ce qui entre, ce qui est mémorisé, et ce qui sort, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Entrées, sorties et état ainsi : Les entrées sont ce que la procédure reçoit ; les sorties sont ce qu'elle produit ; l'état est ce qui change en cours de route.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait entrées, sorties et état ?
2. Quel détail de « un distributeur automatique qui se souvient de votre crédit » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 1.3 La justesse avant l'ingéniosité

# La justesse avant l'ingéniosité

*Une procédure utile doit résoudre le problème promis pour toute entrée autorisée, pas seulement pour un exemple favorable.*

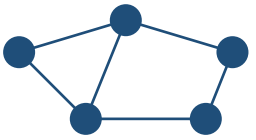


*Figure 1.3. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Une entrée concrète. Pensez à tester des directions sur une route fermée. Cela offre un chemin mémorable vers cette définition : Une procédure utile doit résoudre le problème promis pour toute entrée autorisée, pas seulement pour un exemple favorable. Une fois cette promesse centrale claire, le pseudocode devient la description d'actions familières plutôt qu'un mur de symboles.

Représenter ce problème. Pour la justesse avant l'ingéniosité, l'entrée doit conserver les distinctions nécessaires pour appliquer la règle. Selon l'application, cela peut signifier des positions ordonnées, des voisins liés, des caractéristiques mesurées, des étiquettes, ou des connexions pondérées. La représentation utilisée pour écrire des exemples, des cas limites, et une condition d'arrêt doit omettre les détails décoratifs tout en conservant chaque fait qui pourrait changer la réponse.

Progrès et arrêt. Chaque étape de la justesse avant l'ingéniosité doit rapprocher le processus d'une condition vérifiable. Une trace utile indique l'état courant, la décision qui vient d'être prise, et le travail restant. Si rien de tout cela ne change, la procédure est peut-être bloquée, ou sa règle d'arrêt est incomplète.

## SOUS LE CAPOT | La justesse avant l'ingéniosité

De l'exemple de classe au produit. Pour écrire des exemples, des cas limites, et une condition d'arrêt, les ingénieurs combinent la justesse avant l'ingéniosité avec le stockage, la validation, la surveillance, et des décisions d'interface. L'échelle peut exiger de l'approximation ou du traitement par lots, tandis que les conséquences humaines peuvent exiger des explications, des annulations, et des seuils prudents.

Un test de qualité pratique. Faites varier la taille, l'ordre, et la propreté de l'entrée. Vérifiez non seulement si la justesse avant l'ingéniosité renvoie une réponse, mais si cette réponse reste pertinente pour écrire des exemples, des cas limites, et une condition d'arrêt. Mesurez la mémoire, le délai, et différents types d'erreurs plutôt que de réduire la qualité à un simple taux de réussite.

### Traduction quotidienne

*Pensez à tester des directions sur une route fermée. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
function find_maximum(numbers):
    if length(numbers) == 0:
        report an unsupported case
    best = numbers[0]
    for value in numbers[1:]:
        if value > best:
            best = value
    return best
# check: works for one item, ties, negative numbers, and rejects empty input
# instead of only the friendly example you first imagined
```

## LABORATOIRE VISUEL | La justesse avant l'ingéniosité

### 1. Traduire l'analogie

Esquissez tester des directions sur une route fermée. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de la justesse avant l'ingéniosité. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Comparez un exemple de la justesse avant l'ingéniosité de petite taille à un exemple un million de fois plus grand. Notez quelle action devient le goulot d'étranglement.

### 4. Remettre en question une hypothèse

Retirez une hypothèse de la définition de la justesse avant l'ingéniosité — ordre, connexité, coût non négatif, ou données représentatives — et prédisez l'échec.

### 5. Localiser les choix humains

Dans un système impliquant écrire des exemples, des cas limites, et une condition d'arrêt, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre La justesse avant l'ingéniosité ainsi : Une procédure utile doit résoudre le problème promis pour toute entrée autorisée, pas seulement pour un exemple favorable.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait la justesse avant l'ingéniosité ?
2. Quel détail de « tester des directions sur une route fermée » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 1.4 L'efficacité comme récit de croissance

# L'efficacité comme récit de croissance

*L'efficacité demande comment le travail croît quand l'entrée grandit, sans exiger de mathématiques avancées.*



*Figure 1.4. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Construire l'image mentale. Imaginez servir dix invités contre dix mille. L'efficacité demande comment le travail croît quand l'entrée grandit, sans exiger de mathématiques avancées. La comparaison met en évidence le rôle de l'ordre et du changement, sans prétendre que le mécanisme réel est littéralement la scène quotidienne.

Du monde réel aux données. La scène « servir dix invités contre dix mille » contient plus de détails qu'un programme n'en a besoin. Un concepteur choisit les éléments pertinents pour l'efficacité comme récit de croissance et les encode en valeurs et relations. Si une distinction cruciale se perd durant cette traduction, aucune astuce ultérieure ne pourra la reconstruire de manière fiable.

Retracer le mécanisme. Parcourez un petit exemple de l'efficacité comme récit de croissance et arrêtez-vous après chaque opération. Demandez-vous quelle promesse de la définition est désormais vraie et quels cas restent possibles. Cette trace disciplinée est le moyen le plus sûr de découvrir des cas oubliés, des mises à jour incorrectes, et des boucles qui ne se terminent jamais.

## SOUS LE CAPOT | L'efficacité comme récit de croissance

Où vous pourriez le rencontrer. Les applications impliquant comparer des balayages répétés à la méthode diviser-pour-régner utilisent cette idée parce que l'information sous-jacente a la forme décrite ci-dessus. Les versions en production doivent aussi composer avec le délai, des mesures imparfaites, un comportement changeant, et des objectifs qui se concurrencent.

Limites et modes d'échec. Testez l'efficacité comme récit de croissance avec aucun élément, un seul élément, des éléments égaux, une entrée malformée, et un cas beaucoup plus grand que prévu. Demandez-vous ensuite si la représentation elle-même est juste et fidèle. Des étapes correctes ne peuvent pas compenser des données qui décrivent le mauvais problème.

### Traduction quotidienne

*Pensez à servir dix invités contre dix mille. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
function contains_slow(items, target):     # checks every item
    for value in items:
        if value == target:
            return true
    return false                            # work grows with length(items)

function contains_fast(sorted_items, target):  # binary search
    low, high = 0, length(sorted_items) - 1
    while low <= high:
        mid = (low + high) // 2
        if sorted_items[mid] == target:
            return true
        if sorted_items[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return false                            # work grows with log(length(items))
```

# LABORATOIRE VISUEL | L'efficacité comme récit de croissance

### 1. Traduire l'analogie

Esquissez servir dix invités contre dix mille. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de l'efficacité comme récit de croissance. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Doublez la quantité d'information utilisée par l'efficacité comme récit de croissance. Prédisez quel travail double, lequel pourrait croître plus vite, et lequel reste fixe.

### 4. Remettre en question une hypothèse

Introduisez une information manquante dans l'exemple de servir dix invités contre dix mille. Décidez s'il faut la rejeter, la réparer, l'ignorer, ou la représenter explicitement.

### 5. Localiser les choix humains

Dans un système impliquant comparer des balayages répétés à la méthode diviser-pour-régner, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre L'efficacité comme récit de croissance ainsi : L'efficacité demande comment le travail croît quand l'entrée grandit, sans exiger de mathématiques avancées.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait l'efficacité comme récit de croissance ?
2. Quel détail de « servir dix invités contre dix mille » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

# Chapitre 1 Révision

## Consignes de restitution

Enseignez les idées centrales de les algorithmes sans intimidation à quelqu'un qui n'a jamais programmé. Utilisez les analogies du chapitre, mais nommez aussi l'information exacte mémorisée par une recette pour décider et opposez-l'à l'efficacité comme récit de croissance. Terminez en expliquant comment chaque processus sait quand s'arrêter.

## Carte des connexions

| Concept | Se relie à | Demandez-vous |
|---|---|---|
| Une recette pour décider | transformer un objectif flou en petites étapes vérifiables | Que change quand l'entrée grandit ? |
| Entrées, sorties et état | séparer ce qui entre, ce qui est mémorisé, et ce qui sort | Que change quand l'entrée grandit ? |
| La justesse avant l'ingéniosité | écrire des exemples, des cas limites, et une condition d'arrêt | Que change quand l'entrée grandit ? |
| L'efficacité comme récit de croissance | comparer des balayages répétés à la méthode diviser-pour-régner | Que change quand l'entrée grandit ? |

## À essayer sur papier

Choisissez cinq objets du quotidien qui correspondent à l'idée d'une recette pour décider. Dessinez la représentation, effectuez une opération à la main, et annotez seulement l'information qui a changé. Expliquez ensuite pourquoi l'information inchangée satisfait toujours la promesse centrale du chapitre.

CHAPITRE 2

# Données : valeurs, enregistrements et références

## Dans ce chapitre

**2.1 Valeurs et types.** Un type décrit ce que signifie une valeur et quelles opérations ont un sens pour elle.

**2.2 Enregistrements.** Un enregistrement regroupe plusieurs champs liés pour qu'ils voyagent comme une seule unité significative.

**2.3 Références.** Une référence est un chemin vers une autre valeur plutôt qu'une copie de cette valeur.

**2.4 Identité et égalité.** Deux enregistrements peuvent contenir des informations égales tout en étant des objets différents.

## Boussole du chapitre

En lisant Données : valeurs, enregistrements et références, comparez les quatre idées de ce chapitre : Valeurs et types, Enregistrements, Références, Identité et égalité. Traitez chaque bloc de pseudocode comme une carte de décisions, pas comme un test de programmation.

2.1 Valeurs et types

# Valeurs et types

*Un type décrit ce que signifie une valeur et quelles opérations ont un sens pour elle.*



*Figure 2.1. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Un point de départ intuitif. Commencez par ne pas ajouter une adresse postale à une température. Dans cette scène, remarquez ce qui peut bouger, ce qui sert de limite, et ce qui compte comme réussite. Ces mêmes questions révèlent l'idée centrale de valeurs et types : Un type décrit ce que signifie une valeur et quelles opérations ont un sens pour elle.

Le point de vue de la machine. Un ordinateur ne reçoit ni la métaphore ni son bon sens. Il reçoit une description structurée choisie pour cette tâche. Dans valeurs et types, cette description doit soutenir l'opération impliquée par la définition ; dans distinguer étiquettes, mesures, valeurs de vérité, et collections, elle doit aussi refléter les unités, les valeurs manquantes et le sens de chaque lien.

L'invariant à surveiller. Pendant valeurs et types, un fait doit rester fiable même quand le reste de l'information change. Ce fait peut être une région ordonnée, un ensemble de sommets visités, un meilleur score connu, ou une frontière autour du travail non résolu. Le nommer rend à la fois la procédure et sa condition d'arrêt plus faciles à vérifier.

## SOUS LE CAPOT | Valeurs et types

À l'intérieur d'un système réel. Valeurs et types contribue à distinguer étiquettes, mesures, valeurs de vérité, et collections, mais ce n'est qu'une étape. Un usage déployé de valeurs et types rassemble aussi et vérifie les entrées, applique des règles, gère les échecs, présente les résultats, et observe ce qui se passe ensuite. Les étapes environnantes déterminent souvent si un algorithme correct devient une expérience utile.

Ce qui mérite la suspicion. Un exemple bien propre peut cacher un coût, une ambiguïté, ou une hypothèse fragile. Pour valeurs et types, examinez l'entrée la plus défavorable et toute égalité ou condition limite. Dans distinguer étiquettes, mesures, valeurs de vérité, et collections, demandez-vous aussi quelles erreurs sont les plus conséquentes et quel chemin de récupération existe.

### Traduction quotidienne

*Pensez à ne pas ajouter une adresse postale à une température. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
function safe_add(a, b):
    if type_of(a) != type_of(b):
        report an unsupported case      # e.g. adding a street address to a tempera
ture
    return a + b


is_number(37)     -> true
is_text("37")     -> true
is_boolean(true)  -> true
```

# LABORATOIRE VISUEL | Valeurs et types

### 1. Traduire l'analogie

Esquissez ne pas ajouter une adresse postale à une température. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de valeurs et types. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Imaginez l'exemple de ne pas ajouter une adresse postale à une température grandir jusqu'à ne plus tenir en mémoire. Décidez ce qui pourrait être traité par morceaux.

### 4. Remettre en question une hypothèse

Créez un contre-exemple qui rend un usage naïf de valeurs et types lent, ambigu, ou trompeur.

### 5. Localiser les choix humains

Dans un système impliquant distinguer étiquettes, mesures, valeurs de vérité, et collections, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Valeurs et types ainsi : Un type décrit ce que signifie une valeur et quelles opérations ont un sens pour elle.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait valeurs et types ?
2. Quel détail de « ne pas ajouter une adresse postale à une température » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 2.2 Enregistrements

# Enregistrements

*Un enregistrement regroupe plusieurs champs liés pour qu'ils voyagent comme une seule unité significative.*



*Figure 2.2. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Voir la règle avant la notation. L'image d'une fiche de catalogue de bibliothèque donne à enregistrements une forme concrète. Un enregistrement regroupe plusieurs champs liés pour qu'ils voyagent comme une seule unité significative. Nous garderons la structure utile de la comparaison tout en écartant les détails qui n'appartiennent pas à l'algorithme.

Choisir ce qu'il faut mémoriser. Une implémentation d'enregistrements repose sur un modèle délibérément choisi de la situation. Le meilleur modèle n'est pas le plus détaillé ; c'est la représentation la plus simple qui conserve les décisions que la méthode doit prendre. C'est particulièrement important pour modéliser un contact, un produit, ou un rendez-vous, où un détail superflu peut ajouter du coût ou du bruit.

Suivre un passage complet. Partez d'un état initial valide, appliquez l'action définissant enregistrements, et notez ce que cette action établit. L'étape suivante doit exploiter cette nouvelle connaissance plutôt que de repartir de zéro. Le progrès devient visible quand la partie non résolue rétrécit, quand la frontière avance, ou quand l'estimation s'améliore.

## SOUS LE CAPOT | Enregistrements

Pourquoi cette idée se transpose bien. La même promesse structurelle rend enregistrements utile pour modéliser un contact, un produit, ou un rendez-vous. Les noms des objets peuvent changer, mais l'opération reste reconnaissable. Ce transfert — d'une histoire de surface à une autre — est le signe que l'on a appris l'algorithme plutôt que mémorisé un exemple.

Des compromis, pas de la magie. Enregistrements gagne quelque chose — vitesse, structure, adaptabilité, ou une estimation utile — en s'appuyant sur des hypothèses particulières. Quand ces hypothèses échouent, le résultat peut devenir lent ou trompeur pour modéliser un contact, un produit, ou un rendez-vous. Une implémentation robuste détecte les cas non pris en charge plutôt que d'inventer silencieusement une confiance.

### Traduction quotidienne

*Pensez à une fiche de catalogue de bibliothèque. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
record Contact:
    name: text
    phone: text
    email: text


function make_contact(name, phone, email):
    return Contact(name=name, phone=phone, email=email)


contact = make_contact("Aisha", "555-0100", "aisha@example.com")
contact.phone      # field access, not a search
```

# LABORATOIRE VISUEL | Enregistrements

### 1. Traduire l'analogie

Esquissez une fiche de catalogue de bibliothèque. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative d'enregistrements. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Comparez un cas d'enregistrements presque résolu à un cas délibérément difficile. Expliquez pourquoi leur comportement d'exécution peut différer.

### 4. Remettre en question une hypothèse

Testez enregistrements avec un cas vide, un cas unique, des égalités, et une valeur malformée. Écrivez la réponse la plus sûre pour chacun.

### 5. Localiser les choix humains

Dans un système impliquant modéliser un contact, un produit, ou un rendez-vous, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Enregistrements ainsi : Un enregistrement regroupe plusieurs champs liés pour qu'ils voyagent comme une seule unité significative.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait enregistrements ?
2. Quel détail de « une fiche de catalogue de bibliothèque » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 2.3 Références

# Références

*Une référence est un chemin vers une autre valeur plutôt qu'une copie de cette valeur.*

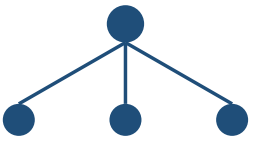


*Figure 2.3. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Une entrée concrète. Pensez à un ticket de vestiaire pour un manteau. Cela offre un chemin mémorable vers cette définition : Une référence est un chemin vers une autre valeur plutôt qu'une copie de cette valeur. Une fois cette promesse centrale claire, le pseudocode devient la description d'actions familières plutôt qu'un mur de symboles.

Représenter ce problème. Pour références, l'entrée doit conserver les distinctions nécessaires pour appliquer la règle. Selon l'application, cela peut signifier des positions ordonnées, des voisins liés, des caractéristiques mesurées, des étiquettes, ou des connexions pondérées. La représentation utilisée pour partager et relier de l'information sans la dupliquer doit omettre les détails décoratifs tout en conservant chaque fait qui pourrait changer la réponse.

Progrès et arrêt. Chaque étape de références doit rapprocher le processus d'une condition vérifiable. Une trace utile indique l'état courant, la décision qui vient d'être prise, et le travail restant. Si rien de tout cela ne change, la procédure est peut-être bloquée, ou sa règle d'arrêt est incomplète.

## SOUS LE CAPOT | Références

De l'exemple de classe au produit. Pour partager et relier de l'information sans la dupliquer, les ingénieurs combinent références avec le stockage, la validation, la surveillance, et des décisions d'interface. L'échelle peut exiger de l'approximation ou du traitement par lots, tandis que les conséquences humaines peuvent exiger des explications, des annulations, et des seuils prudents.

Un test de qualité pratique. Faites varier la taille, l'ordre, et la propreté de l'entrée. Vérifiez non seulement si références renvoie une réponse, mais si cette réponse reste pertinente pour partager et relier de l'information sans la dupliquer. Mesurez la mémoire, le délai, et différents types d'erreurs plutôt que de réduire la qualité à un simple taux de réussite.

### Traduction quotidienne

*Pensez à un ticket de vestiaire pour un manteau. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
locker_a = Contact("Aisha", "555-0100")
locker_b = locker_a          # locker_b now refers to the same record
locker_b.phone = "555-0199"

print(locker_a.phone)         # "555-0199" -- both names point to one record
# a copy would have left locker_a.phone unchanged
```

## LABORATOIRE VISUEL | Références

### 1. Traduire l'analogie

Esquissez un ticket de vestiaire pour un manteau. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de références. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Comparez un exemple de références de petite taille à un exemple un million de fois plus grand. Notez quelle action devient le goulot d'étranglement.

### 4. Remettre en question une hypothèse

Retirez une hypothèse de la définition de références — ordre, connexité, coût non négatif, ou données représentatives — et prédisez l'échec.

### 5. Localiser les choix humains

Dans un système impliquant partager et relier de l'information sans la dupliquer, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Références ainsi : Une référence est un chemin vers une autre valeur plutôt qu'une copie de cette valeur.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait références ?
2. Quel détail de « un ticket de vestiaire pour un manteau » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

2.4 Identité et égalité

# Identité et égalité

*Deux enregistrements peuvent contenir des informations égales tout en étant des objets différents.*

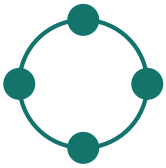


*Figure 2.4. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Construire l'image mentale. Imaginez deux clés identiques taillées d'un même modèle. Deux enregistrements peuvent contenir des informations égales tout en étant des objets différents. La comparaison met en évidence le rôle de l'ordre et du changement, sans prétendre que le mécanisme réel est littéralement la scène quotidienne.

Du monde réel aux données. La scène « deux clés identiques taillées d'un même modèle » contient plus de détails qu'un programme n'en a besoin. Un concepteur choisit les éléments pertinents pour identité et égalité et les encode en valeurs et relations. Si une distinction cruciale se perd durant cette traduction, aucune astuce ultérieure ne pourra la reconstruire de manière fiable.

Retracer le mécanisme. Parcourez un petit exemple d'identité et égalité et arrêtez-vous après chaque opération. Demandez-vous quelle promesse de la définition est désormais vraie et quels cas restent possibles. Cette trace disciplinée est le moyen le plus sûr de découvrir des cas oubliés, des mises à jour incorrectes, et des boucles qui ne se terminent jamais.

## SOUS LE CAPOT | Identité et égalité

Où vous pourriez le rencontrer. Les applications impliquant raisonner sur les copies, les alias, et les changements utilisent cette idée parce que l'information sous-jacente a la forme décrite ci-dessus. Les versions en production doivent aussi composer avec le délai, des mesures imparfaites, un comportement changeant, et des objectifs qui se concurrencent.

Limites et modes d'échec. Testez identité et égalité avec aucun élément, un seul élément, des éléments égaux, une entrée malformée, et un cas beaucoup plus grand que prévu. Demandez-vous ensuite si la représentation elle-même est juste et fidèle. Des étapes correctes ne peuvent pas compenser des données qui décrivent le mauvais problème.

### Traduction quotidienne

*Pensez à deux clés identiques taillées d'un même modèle. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
a = Contact("Aisha", "555-0100")
b = Contact("Aisha", "555-0100")
c = a


a == b        # true  -- equal contents
a is b        # false -- two distinct records
a is c        # true  -- the same record, reached two ways
```

## LABORATOIRE VISUEL | Identité et égalité

### 1. Traduire l'analogie

Esquissez deux clés identiques taillées d'un même modèle. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative d'identité et égalité. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Doublez la quantité d'information utilisée par identité et égalité. Prédisez quel travail double, lequel pourrait croître plus vite, et lequel reste fixe.

### 4. Remettre en question une hypothèse

Introduisez une information manquante dans l'exemple de deux clés identiques taillées d'un même modèle. Décidez s'il faut la rejeter, la réparer, l'ignorer, ou la représenter explicitement.

### 5. Localiser les choix humains

Dans un système impliquant raisonner sur les copies, les alias, et les changements, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Identité et égalité ainsi : Deux enregistrements peuvent contenir des informations égales tout en étant des objets différents.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait identité et égalité ?
2. Quel détail de « deux clés identiques taillées d'un même modèle » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

# Chapitre 2 Révision

## Consignes de restitution

Enseignez les idées centrales de données : valeurs, enregistrements et références à quelqu'un qui n'a jamais programmé. Utilisez les analogies du chapitre, mais nommez aussi l'information exacte mémorisée par valeurs et types et opposez-l'à identité et égalité. Terminez en expliquant comment chaque processus sait quand s'arrêter.

## Carte des connexions

| Concept | Se relie à | Demandez-vous |
|---|---|---|
| Valeurs et types | distinguer étiquettes, mesures, valeurs de vérité, et collections | Que change quand l'entrée grandit ? |
| Enregistrements | modéliser un contact, un produit, ou un rendez-vous | Que change quand l'entrée grandit ? |
| Références | partager et relier de l'information sans la dupliquer | Que change quand l'entrée grandit ? |
| Identité et égalité | raisonner sur les copies, les alias, et les changements | Que change quand l'entrée grandit ? |

## À essayer sur papier

Choisissez cinq objets du quotidien qui correspondent à l'idée d'enregistrements. Dessinez la représentation, effectuez une opération à la main, et annotez seulement l'information qui a changé. Expliquez ensuite pourquoi l'information inchangée satisfait toujours la promesse centrale du chapitre.

CHAPITRE 3

# Tableaux et listes dynamiques

## Dans ce chapitre

**3.1 Une rangée de casiers numérotés.** Un tableau stocke des éléments dans l'ordre et donne à chaque position un indice.

**3.2 Insertion et suppression.** Faire de la place au milieu peut exiger de décaler chaque élément suivant.

**3.3 Grandir au-delà de la capacité.** Un tableau dynamique se déplace parfois vers un espace plus grand pour que la plupart des ajouts restent peu coûteux.

**3.4 Tranches et vues.** Une tranche décrit une région contiguë ; certains systèmes la copient tandis que d'autres partagent le stockage original.

## Boussole du chapitre

En lisant Tableaux et listes dynamiques, comparez les quatre idées de ce chapitre : Une rangée de casiers numérotés, Insertion et suppression, Grandir au-delà de la capacité, Tranches et vues. Traitez chaque bloc de pseudocode comme une carte de décisions, pas comme un test de programmation.

### 3.1 Une rangée de casiers numérotés

# Une rangée de casiers numérotés

*Un tableau stocke des éléments dans l'ordre et donne à chaque position un indice.*

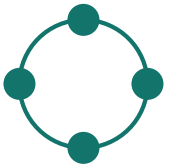


*Figure 3.1. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Un point de départ intuitif. Commencez par une rangée de casiers numérotés. Dans cette scène, remarquez ce qui peut bouger, ce qui sert de limite, et ce qui compte comme réussite. Ces mêmes questions révèlent l'idée centrale d'une rangée de casiers numérotés : Un tableau stocke des éléments dans l'ordre et donne à chaque position un indice.

Le point de vue de la machine. Un ordinateur ne reçoit ni la métaphore ni son bon sens. Il reçoit une description structurée choisie pour cette tâche. Dans une rangée de casiers numérotés, cette description doit soutenir l'opération impliquée par la définition ; dans un accès direct rapide et un stockage compact, elle doit aussi refléter les unités, les valeurs manquantes et le sens de chaque lien.

L'invariant à surveiller. Pendant une rangée de casiers numérotés, un fait doit rester fiable même quand le reste de l'information change. Ce fait peut être une région ordonnée, un ensemble de sommets visités, un meilleur score connu, ou une frontière autour du travail non résolu. Le nommer rend à la fois la procédure et sa condition d'arrêt plus faciles à vérifier.

## SOUS LE CAPOT | Une rangée de casiers numérotés

À l'intérieur d'un système réel. Une rangée de casiers numérotés contribue à un accès direct rapide et un stockage compact, mais ce n'est qu'une étape. Un usage déployé d'une rangée de casiers numérotés rassemble aussi et vérifie les entrées, applique des règles, gère les échecs, présente les résultats, et observe ce qui se passe ensuite. Les étapes environnantes déterminent souvent si un algorithme correct devient une expérience utile.

Ce qui mérite la suspicion. Un exemple bien propre peut cacher un coût, une ambiguïté, ou une hypothèse fragile. Pour une rangée de casiers numérotés, examinez l'entrée la plus défavorable et toute égalité ou condition limite. Dans un accès direct rapide et un stockage compact, demandez-vous aussi quelles erreurs sont les plus conséquentes et quel chemin de récupération existe.

### Traduction quotidienne

*Pensez à une rangée de casiers numérotés. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
locker = ["tea", "cups", "kettle", "napkins"]


locker[0]          # "tea"      -- direct access by index, O(1)
locker[2] = "pot"  # replace in place, O(1)
length(locker)     # 4
```

# LABORATOIRE VISUEL | Une rangée de casiers numérotés

### 1. Traduire l'analogie

Esquissez une rangée de casiers numérotés. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative d'une rangée de casiers numérotés. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Imaginez l'exemple d'une rangée de casiers numérotés grandir jusqu'à ne plus tenir en mémoire. Décidez ce qui pourrait être traité par morceaux.

### 4. Remettre en question une hypothèse

Créez un contre-exemple qui rend un usage naïf d'une rangée de casiers numérotés lent, ambigu, ou trompeur.

### 5. Localiser les choix humains

Dans un système impliquant un accès direct rapide et un stockage compact, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Une rangée de casiers numérotés ainsi : Un tableau stocke des éléments dans l'ordre et donne à chaque position un indice.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait une rangée de casiers numérotés ?
2. Quel détail de « une rangée de casiers numérotés » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 3.2 Insertion et suppression

# Insertion et suppression

*Faire de la place au milieu peut exiger de décaler chaque élément suivant.*

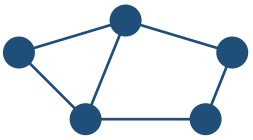


*Figure 3.2. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Voir la règle avant la notation. L'image d'ajouter une personne au milieu d'une rangée de théâtre donne à insertion et suppression une forme concrète. Faire de la place au milieu peut exiger de décaler chaque élément suivant. Nous garderons la structure utile de la comparaison tout en écartant les détails qui n'appartiennent pas à l'algorithme.

Choisir ce qu'il faut mémoriser. Une implémentation d'insertion et suppression repose sur un modèle délibérément choisi de la situation. Le meilleur modèle n'est pas le plus détaillé ; c'est la représentation la plus simple qui conserve les décisions que la méthode doit prendre. C'est particulièrement important pour comprendre pourquoi les changements de position peuvent coûter cher, où un détail superflu peut ajouter du coût ou du bruit.

Suivre un passage complet. Partez d'un état initial valide, appliquez l'action définissant insertion et suppression, et notez ce que cette action établit. L'étape suivante doit exploiter cette nouvelle connaissance plutôt que de repartir de zéro. Le progrès devient visible quand la partie non résolue rétrécit, quand la frontière avance, ou quand l'estimation s'améliore.

## SOUS LE CAPOT | Insertion et suppression

Pourquoi cette idée se transpose bien. La même promesse structurelle rend insertion et suppression utile pour comprendre pourquoi les changements de position peuvent coûter cher. Les noms des objets peuvent changer, mais l'opération reste reconnaissable. Ce transfert — d'une histoire de surface à une autre — est le signe que l'on a appris l'algorithme plutôt que mémorisé un exemple.

Des compromis, pas de la magie. Insertion et suppression gagne quelque chose — vitesse, structure, adaptabilité, ou une estimation utile — en s'appuyant sur des hypothèses particulières. Quand ces hypothèses échouent, le résultat peut devenir lent ou trompeur pour comprendre pourquoi les changements de position peuvent coûter cher. Une implémentation robuste détecte les cas non pris en charge plutôt que d'inventer silencieusement une confiance.

### Traduction quotidienne

*Pensez à ajouter une personne au milieu d'une rangée de théâtre. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
function insert_at(items, index, value):
    items.append(placeholder)
    for i from length(items) - 1 down to index + 1:
        items[i] = items[i - 1]      # shift right to make room
    items[index] = value
    return items                      # O(n) in the worst case

function delete_at(items, index):
    for i from index to length(items) - 2:
        items[i] = items[i + 1]      # shift left to close the gap
    items.remove_last()
    return items                      # O(n) in the worst case
```

# LABORATOIRE VISUEL | Insertion et suppression

### 1. Traduire l'analogie

Esquissez ajouter une personne au milieu d'une rangée de théâtre. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative d'insertion et suppression. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Comparez un cas d'insertion et suppression presque résolu à un cas délibérément difficile. Expliquez pourquoi leur comportement d'exécution peut différer.

### 4. Remettre en question une hypothèse

Testez insertion et suppression avec un cas vide, un cas unique, des égalités, et une valeur malformée. Écrivez la réponse la plus sûre pour chacun.

### 5. Localiser les choix humains

Dans un système impliquant comprendre pourquoi les changements de position peuvent coûter cher, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Insertion et suppression ainsi : Faire de la place au milieu peut exiger de décaler chaque élément suivant.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait insertion et suppression ?
2. Quel détail de « ajouter une personne au milieu d'une rangée de théâtre » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 3.3 Grandir au-delà de la capacité

# Grandir au-delà de la capacité

*Un tableau dynamique se déplace parfois vers un espace plus grand pour que la plupart des ajouts restent peu coûteux.*



*Figure 3.3. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Une entrée concrète. Pensez à déménager vers une étagère plus grande avant que chaque étagère soit pleine. Cela offre un chemin mémorable vers cette définition : Un tableau dynamique se déplace parfois vers un espace plus grand pour que la plupart des ajouts restent peu coûteux. Une fois cette promesse centrale claire, le pseudocode devient la description d'actions familières plutôt qu'un mur de symboles.

Représenter ce problème. Pour grandir au-delà de la capacité, l'entrée doit conserver les distinctions nécessaires pour appliquer la règle. Selon l'application, cela peut signifier des positions ordonnées, des voisins liés, des caractéristiques mesurées, des étiquettes, ou des connexions pondérées. La représentation utilisée pour équilibrer la place de réserve et le coût de la copie doit omettre les détails décoratifs tout en conservant chaque fait qui pourrait changer la réponse.

Progrès et arrêt. Chaque étape de grandir au-delà de la capacité doit rapprocher le processus d'une condition vérifiable. Une trace utile indique l'état courant, la décision qui vient d'être prise, et le travail restant. Si rien de tout cela ne change, la procédure est peut-être bloquée, ou sa règle d'arrêt est incomplète.

## SOUS LE CAPOT | Grandir au-delà de la capacité

De l'exemple de classe au produit. Pour équilibrer la place de réserve et le coût de la copie, les ingénieurs combinent grandir au-delà de la capacité avec le stockage, la validation, la surveillance, et des décisions d'interface. L'échelle peut exiger de l'approximation ou du traitement par lots, tandis que les conséquences humaines peuvent exiger des explications, des annulations, et des seuils prudents.

Un test de qualité pratique. Faites varier la taille, l'ordre, et la propreté de l'entrée. Vérifiez non seulement si grandir au-delà de la capacité renvoie une réponse, mais si cette réponse reste pertinente pour équilibrer la place de réserve et le coût de la copie. Mesurez la mémoire, le délai, et différents types d'erreurs plutôt que de réduire la qualité à un simple taux de réussite.

### Traduction quotidienne

*Pensez à déménager vers une étagère plus grande avant que chaque étagère soit pleine. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
function append(dynamic_array, value):
    if dynamic_array.count == dynamic_array.capacity:
        new_capacity = dynamic_array.capacity * 2      # double, don't add one
        new_storage = allocate(new_capacity)
        copy dynamic_array.storage into new_storage
        dynamic_array.storage = new_storage
        dynamic_array.capacity = new_capacity
    dynamic_array.storage[dynamic_array.count] = value
    dynamic_array.count += 1
# occasional O(n) copies, but appends are O(1) on average
```

# LABORATOIRE VISUEL | Grandir au-delà de la capacité

### 1. Traduire l'analogie

Esquissez déménager vers une étagère plus grande avant que chaque étagère soit pleine. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de grandir au-delà de la capacité. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Comparez un exemple de grandir au-delà de la capacité de petite taille à un exemple un million de fois plus grand. Notez quelle action devient le goulot d'étranglement.

### 4. Remettre en question une hypothèse

Retirez une hypothèse de la définition de grandir au-delà de la capacité — ordre, connexité, coût non négatif, ou données représentatives — et prédisez l'échec.

### 5. Localiser les choix humains

Dans un système impliquant équilibrer la place de réserve et le coût de la copie, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Grandir au-delà de la capacité ainsi : Un tableau dynamique se déplace parfois vers un espace plus grand pour que la plupart des ajouts restent peu coûteux.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait grandir au-delà de la capacité ?
2. Quel détail de « déménager vers une étagère plus grande avant que chaque étagère soit pleine » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 3.4 Tranches et vues

# Tranches et vues

*Une tranche décrit une région contiguë ; certains systèmes la copient tandis que d'autres partagent le stockage original.*



*Figure 3.4. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Construire l'image mentale. Imaginez regarder par une fenêtre une partie d'une fresque. Une tranche décrit une région contiguë ; certains systèmes la copient tandis que d'autres partagent le stockage original. La comparaison met en évidence le rôle de l'ordre et du changement, sans prétendre que le mécanisme réel est littéralement la scène quotidienne.

Du monde réel aux données. La scène « regarder par une fenêtre une partie d'une fresque » contient plus de détails qu'un programme n'en a besoin. Un concepteur choisit les éléments pertinents pour tranches et vues et les encode en valeurs et relations. Si une distinction cruciale se perd durant cette traduction, aucune astuce ultérieure ne pourra la reconstruire de manière fiable.

Retracer le mécanisme. Parcourez un petit exemple de tranches et vues et arrêtez-vous après chaque opération. Demandez-vous quelle promesse de la définition est désormais vraie et quels cas restent possibles. Cette trace disciplinée est le moyen le plus sûr de découvrir des cas oubliés, des mises à jour incorrectes, et des boucles qui ne se terminent jamais.

## SOUS LE CAPOT | Tranches et vues

Où vous pourriez le rencontrer. Les applications impliquant éviter des changements partagés surprenants utilisent cette idée parce que l'information sous-jacente a la forme décrite ci-dessus. Les versions en production doivent aussi composer avec le délai, des mesures imparfaites, un comportement changeant, et des objectifs qui se concurrencent.

Limites et modes d'échec. Testez tranches et vues avec aucun élément, un seul élément, des éléments égaux, une entrée malformée, et un cas beaucoup plus grand que prévu. Demandez-vous ensuite si la représentation elle-même est juste et fidèle. Des étapes correctes ne peuvent pas compenser des données qui décrivent le mauvais problème.

### Traduction quotidienne

*Pensez à regarder par une fenêtre une partie d'une fresque. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
items = [10, 20, 30, 40, 50]


items[1:4]        # [20, 30, 40] -- a new list, indices 1 up to (not including) 4
items[:2]         # [10, 20]
items[-1]         # 50 -- last element
# a view can reference the same storage instead of copying;
# check which behavior a language gives you before mutating one side
```

# LABORATOIRE VISUEL | Tranches et vues

### 1. Traduire l'analogie

Esquissez regarder par une fenêtre une partie d'une fresque. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de tranches et vues. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Doublez la quantité d'information utilisée par tranches et vues. Prédisez quel travail double, lequel pourrait croître plus vite, et lequel reste fixe.

### 4. Remettre en question une hypothèse

Introduisez une information manquante dans l'exemple de regarder par une fenêtre une partie d'une fresque. Décidez s'il faut la rejeter, la réparer, l'ignorer, ou la représenter explicitement.

### 5. Localiser les choix humains

Dans un système impliquant éviter des changements partagés surprenants, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Tranches et vues ainsi : Une tranche décrit une région contiguë ; certains systèmes la copient tandis que d'autres partagent le stockage original.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait tranches et vues ?
2. Quel détail de « regarder par une fenêtre une partie d'une fresque » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

# Chapitre 3 Révision

## Consignes de restitution

Enseignez les idées centrales de tableaux et listes dynamiques à quelqu'un qui n'a jamais programmé. Utilisez les analogies du chapitre, mais nommez aussi l'information exacte mémorisée par une rangée de casiers numérotés et opposez-l'à tranches et vues. Terminez en expliquant comment chaque processus sait quand s'arrêter.

## Carte des connexions

| Concept | Se relie à | Demandez-vous |
|---|---|---|
| Une rangée de casiers numérotés | un accès direct rapide et un stockage compact | Que change quand l'entrée grandit ? |
| Insertion et suppression | comprendre pourquoi les changements de position peuvent coûter cher | Que change quand l'entrée grandit ? |
| Grandir au-delà de la capacité | équilibrer la place de réserve et le coût de la copie | Que change quand l'entrée grandit ? |
| Tranches et vues | éviter des changements partagés surprenants | Que change quand l'entrée grandit ? |

## À essayer sur papier

Choisissez cinq objets du quotidien qui correspondent à l'idée de grandir au-delà de la capacité. Dessinez la représentation, effectuez une opération à la main, et annotez seulement l'information qui a changé. Expliquez ensuite pourquoi l'information inchangée satisfait toujours la promesse centrale du chapitre.

CHAPITRE 4

# Listes chaînées

## Dans ce chapitre

**4.1 Suivre la flèche suivante.** Chaque nœud stocke une valeur et un lien vers le nœud suivant.

**4.2 Simplement ou doublement chaînée.** Un second lien vers l'arrière facilite le parcours inverse et la suppression, au prix d'un espace supplémentaire.

**4.3 Pourquoi l'accès aléatoire est lent.** Pour atteindre le centième nœud, une liste chaînée suit les quatre-vingt-dix-neuf premiers liens.

**4.4 Sentinelles.** Un nœud limite factice supprime des cas particuliers gênants au début ou à la fin.

## Boussole du chapitre

En lisant Listes chaînées, comparez les quatre idées de ce chapitre : Suivre la flèche suivante, Simplement ou doublement chaînée, Pourquoi l'accès aléatoire est lent, Sentinelles. Traitez chaque bloc de pseudocode comme une carte de décisions, pas comme un test de programmation.

4.1 Suivre la flèche suivante

# Suivre la flèche suivante

*Chaque nœud stocke une valeur et un lien vers le nœud suivant.*



*Figure 4.1. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Un point de départ intuitif. Commencez par une chasse au trésor faite d'indices. Dans cette scène, remarquez ce qui peut bouger, ce qui sert de limite, et ce qui compte comme réussite. Ces mêmes questions révèlent l'idée centrale de suivre la flèche suivante : Chaque nœud stocke une valeur et un lien vers le nœud suivant.

Le point de vue de la machine. Un ordinateur ne reçoit ni la métaphore ni son bon sens. Il reçoit une description structurée choisie pour cette tâche. Dans suivre la flèche suivante, cette description doit soutenir l'opération impliquée par la définition ; dans une insertion peu coûteuse quand l'emplacement est déjà connu, elle doit aussi refléter les unités, les valeurs manquantes et le sens de chaque lien.

L'invariant à surveiller. Pendant suivre la flèche suivante, un fait doit rester fiable même quand le reste de l'information change. Ce fait peut être une région ordonnée, un ensemble de sommets visités, un meilleur score connu, ou une frontière autour du travail non résolu. Le nommer rend à la fois la procédure et sa condition d'arrêt plus faciles à vérifier.

## SOUS LE CAPOT | Suivre la flèche suivante

À l'intérieur d'un système réel. Suivre la flèche suivante contribue à une insertion peu coûteuse quand l'emplacement est déjà connu, mais ce n'est qu'une étape. Un usage déployé de suivre la flèche suivante rassemble aussi et vérifie les entrées, applique des règles, gère les échecs, présente les résultats, et observe ce qui se passe ensuite. Les étapes environnantes déterminent souvent si un algorithme correct devient une expérience utile.

Ce qui mérite la suspicion. Un exemple bien propre peut cacher un coût, une ambiguïté, ou une hypothèse fragile. Pour suivre la flèche suivante, examinez l'entrée la plus défavorable et toute égalité ou condition limite. Dans une insertion peu coûteuse quand l'emplacement est déjà connu, demandez-vous aussi quelles erreurs sont les plus conséquentes et quel chemin de récupération existe.

### Traduction quotidienne

*Pensez à une chasse au trésor faite d'indices. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
function traverse(head):
    node = head
    while node is not null:
        visit(node.value)
        node = node.next        # follow the arrow, one hop at a time
# no index arithmetic -- just "is there a next node?"
```

# LABORATOIRE VISUEL | Suivre la flèche suivante

### 1. Traduire l'analogie

Esquissez une chasse au trésor faite d'indices. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de suivre la flèche suivante. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Imaginez l'exemple d'une chasse au trésor faite d'indices grandir jusqu'à ne plus tenir en mémoire. Décidez ce qui pourrait être traité par morceaux.

### 4. Remettre en question une hypothèse

Créez un contre-exemple qui rend un usage naïf de suivre la flèche suivante lent, ambigu, ou trompeur.

### 5. Localiser les choix humains

Dans un système impliquant une insertion peu coûteuse quand l'emplacement est déjà connu, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Suivre la flèche suivante ainsi : Chaque nœud stocke une valeur et un lien vers le nœud suivant.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait suivre la flèche suivante ?
2. Quel détail de « une chasse au trésor faite d'indices » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 4.2 Simplement ou doublement chaînée

# Simplement ou doublement chaînée

*Un second lien vers l'arrière facilite le parcours inverse et la suppression, au prix d'un espace supplémentaire.*

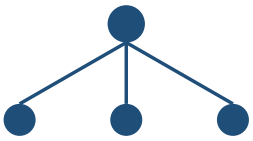


*Figure 4.2. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Voir la règle avant la notation. L'image de des wagons de train attelés dans une ou deux directions donne à simplement ou doublement chaînée une forme concrète. Un second lien vers l'arrière facilite le parcours inverse et la suppression, au prix d'un espace supplémentaire. Nous garderons la structure utile de la comparaison tout en écartant les détails qui n'appartiennent pas à l'algorithme.

Choisir ce qu'il faut mémoriser. Une implémentation de simplement ou doublement chaînée repose sur un modèle délibérément choisi de la situation. Le meilleur modèle n'est pas le plus détaillé ; c'est la représentation la plus simple qui conserve les décisions que la méthode doit prendre. C'est particulièrement important pour choisir la flexibilité de navigation, où un détail superflu peut ajouter du coût ou du bruit.

Suivre un passage complet. Partez d'un état initial valide, appliquez l'action définissant simplement ou doublement chaînée, et notez ce que cette action établit. L'étape suivante doit exploiter cette nouvelle connaissance plutôt que de repartir de zéro. Le progrès devient visible quand la partie non résolue rétrécit, quand la frontière avance, ou quand l'estimation s'améliore.

## SOUS LE CAPOT | Simplement ou doublement chaînée

Pourquoi cette idée se transpose bien. La même promesse structurelle rend simplement ou doublement chaînée utile pour choisir la flexibilité de navigation. Les noms des objets peuvent changer, mais l'opération reste reconnaissable. Ce transfert — d'une histoire de surface à une autre — est le signe que l'on a appris l'algorithme plutôt que mémorisé un exemple.

Des compromis, pas de la magie. Simplement ou doublement chaînée gagne quelque chose — vitesse, structure, adaptabilité, ou une estimation utile — en s'appuyant sur des hypothèses particulières. Quand ces hypothèses échouent, le résultat peut devenir lent ou trompeur pour choisir la flexibilité de navigation. Une implémentation robuste détecte les cas non pris en charge plutôt que d'inventer silencieusement une confiance.

### Traduction quotidienne

*Pensez à des wagons de train attelés dans une ou deux directions. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
function insert_after(node, value):        # singly linked
    new_node = Node(value)
    new_node.next = node.next
    node.next = new_node                    # O(1) once you hold the node

function insert_after_doubly(node, value):  # doubly linked
    new_node = Node(value)
    new_node.next = node.next
    new_node.prev = node
    if node.next is not null:
        node.next.prev = new_node
    node.next = new_node                    # can also walk backward via .prev
```

# LABORATOIRE VISUEL | Simplement ou doublement chaînée

### 1. Traduire l'analogie

Esquissez des wagons de train attelés dans une ou deux directions. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de simplement ou doublement chaînée. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Comparez un cas de simplement ou doublement chaînée presque résolu à un cas délibérément difficile. Expliquez pourquoi leur comportement d'exécution peut différer.

### 4. Remettre en question une hypothèse

Testez simplement ou doublement chaînée avec un cas vide, un cas unique, des égalités, et une valeur malformée. Écrivez la réponse la plus sûre pour chacun.

### 5. Localiser les choix humains

Dans un système impliquant choisir la flexibilité de navigation, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Simplement ou doublement chaînée ainsi : Un second lien vers l'arrière facilite le parcours inverse et la suppression, au prix d'un espace supplémentaire.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait simplement ou doublement chaînée ?
2. Quel détail de « des wagons de train attelés dans une ou deux directions » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 4.3 Pourquoi l'accès aléatoire est lent

# Pourquoi l'accès aléatoire est lent

*Pour atteindre le centième nœud, une liste chaînée suit les quatre-vingt-dix-neuf premiers liens.*

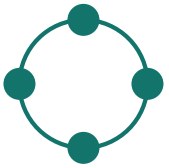


*Figure 4.3. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Une entrée concrète. Pensez à aller de porte en porte sans numéros de maison. Cela offre un chemin mémorable vers cette définition : Pour atteindre le centième nœud, une liste chaînée suit les quatre-vingt-dix-neuf premiers liens. Une fois cette promesse centrale claire, le pseudocode devient la description d'actions familières plutôt qu'un mur de symboles.

Représenter ce problème. Pour pourquoi l'accès aléatoire est lent, l'entrée doit conserver les distinctions nécessaires pour appliquer la règle. Selon l'application, cela peut signifier des positions ordonnées, des voisins liés, des caractéristiques mesurées, des étiquettes, ou des connexions pondérées. La représentation utilisée pour reconnaître le compromis entre l'accès et la modification doit omettre les détails décoratifs tout en conservant chaque fait qui pourrait changer la réponse.

Progrès et arrêt. Chaque étape de pourquoi l'accès aléatoire est lent doit rapprocher le processus d'une condition vérifiable. Une trace utile indique l'état courant, la décision qui vient d'être prise, et le travail restant. Si rien de tout cela ne change, la procédure est peut-être bloquée, ou sa règle d'arrêt est incomplète.

## SOUS LE CAPOT | Pourquoi l'accès aléatoire est lent

De l'exemple de classe au produit. Pour reconnaître le compromis entre l'accès et la modification, les ingénieurs combinent pourquoi l'accès aléatoire est lent avec le stockage, la validation, la surveillance, et des décisions d'interface. L'échelle peut exiger de l'approximation ou du traitement par lots, tandis que les conséquences humaines peuvent exiger des explications, des annulations, et des seuils prudents.

Un test de qualité pratique. Faites varier la taille, l'ordre, et la propreté de l'entrée. Vérifiez non seulement si pourquoi l'accès aléatoire est lent renvoie une réponse, mais si cette réponse reste pertinente pour reconnaître le compromis entre l'accès et la modification. Mesurez la mémoire, le délai, et différents types d'erreurs plutôt que de réduire la qualité à un simple taux de réussite.

### Traduction quotidienne

*Pensez à aller de porte en porte sans numéros de maison. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
function get_at(head, index):
    node = head
    steps = 0
    while steps < index:
        node = node.next        # must hop node by node
        steps += 1
    return node.value            # O(index) -- unlike array[index], which is O(1)
```

# LABORATOIRE VISUEL | Pourquoi l'accès aléatoire est lent

### 1. Traduire l'analogie

Esquissez aller de porte en porte sans numéros de maison. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de pourquoi l'accès aléatoire est lent. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Comparez un exemple de pourquoi l'accès aléatoire est lent de petite taille à un exemple un million de fois plus grand. Notez quelle action devient le goulot d'étranglement.

### 4. Remettre en question une hypothèse

Retirez une hypothèse de la définition de pourquoi l'accès aléatoire est lent — ordre, connexité, coût non négatif, ou données représentatives — et prédisez l'échec.

### 5. Localiser les choix humains

Dans un système impliquant reconnaître le compromis entre l'accès et la modification, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Pourquoi l'accès aléatoire est lent ainsi : Pour atteindre le centième nœud, une liste chaînée suit les quatre-vingt-dix-neuf premiers liens.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait pourquoi l'accès aléatoire est lent ?
2. Quel détail de « aller de porte en porte sans numéros de maison » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

### 4.4 Sentinelles

# Sentinelles

*Un nœud limite factice supprime des cas particuliers gênants au début ou à la fin.*

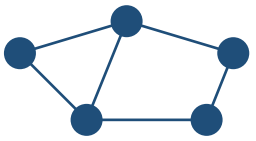


*Figure 4.4. Un rythme visuel pour l'idée : entrée, relation, changement, et résultat.*

Construire l'image mentale. Imaginez une butée de livres qui n'est pas un livre. Un nœud limite factice supprime des cas particuliers gênants au début ou à la fin. La comparaison met en évidence le rôle de l'ordre et du changement, sans prétendre que le mécanisme réel est littéralement la scène quotidienne.

Du monde réel aux données. La scène « une butée de livres qui n'est pas un livre » contient plus de détails qu'un programme n'en a besoin. Un concepteur choisit les éléments pertinents pour sentinelles et les encode en valeurs et relations. Si une distinction cruciale se perd durant cette traduction, aucune astuce ultérieure ne pourra la reconstruire de manière fiable.

Retracer le mécanisme. Parcourez un petit exemple de sentinelles et arrêtez-vous après chaque opération. Demandez-vous quelle promesse de la définition est désormais vraie et quels cas restent possibles. Cette trace disciplinée est le moyen le plus sûr de découvrir des cas oubliés, des mises à jour incorrectes, et des boucles qui ne se terminent jamais.

## SOUS LE CAPOT | Sentinelles

Où vous pourriez le rencontrer. Les applications impliquant simplifier un code autrement fragile utilisent cette idée parce que l'information sous-jacente a la forme décrite ci-dessus. Les versions en production doivent aussi composer avec le délai, des mesures imparfaites, un comportement changeant, et des objectifs qui se concurrencent.

Limites et modes d'échec. Testez sentinelles avec aucun élément, un seul élément, des éléments égaux, une entrée malformée, et un cas beaucoup plus grand que prévu. Demandez-vous ensuite si la représentation elle-même est juste et fidèle. Des étapes correctes ne peuvent pas compenser des données qui décrivent le mauvais problème.

### Traduction quotidienne

*Pensez à une butée de livres qui n'est pas un livre. Utilisez cette image pour retenir la règle, puis revenez à la définition exacte quand les détails comptent.*

### Pseudocode à la Python

```
sentinel = Node(value=None)      # a dummy node that is never real data
sentinel.next = sentinel         # empty list points to itself


function is_empty(list):
    return list.sentinel.next is list.sentinel
# insert/delete code no longer needs a special case for "the list is empty"
```

## LABORATOIRE VISUEL | Sentinelles

### 1. Traduire l'analogie

Esquissez une butée de livres qui n'est pas un livre. Identifiez les objets qui représentent les données stockées, les relations, l'état de travail, et le résultat final.

### 2. Retracer une étape décisive

Effectuez une opération significative de sentinelles. Notez ce qui a été lu, ce qui a changé, et quelle promesse est devenue vraie.

### 3. Examiner la croissance

Doublez la quantité d'information utilisée par sentinelles. Prédisez quel travail double, lequel pourrait croître plus vite, et lequel reste fixe.

### 4. Remettre en question une hypothèse

Introduisez une information manquante dans l'exemple d'une butée de livres qui n'est pas un livre. Décidez s'il faut la rejeter, la réparer, l'ignorer, ou la représenter explicitement.

### 5. Localiser les choix humains

Dans un système impliquant simplifier un code autrement fragile, identifiez une décision automatisée, un choix de conception ou de politique, et un résultat qu'une personne devrait pouvoir remettre en question.

### Expliquer sans jargon

On peut comprendre Sentinelles ainsi : Un nœud limite factice supprime des cas particuliers gênants au début ou à la fin.

### Vérification rapide

1. Avec vos propres mots, quelle promesse fait sentinelles ?
2. Quel détail de « une butée de livres qui n'est pas un livre » correspond à l'état qui change ?
3. Donnez une situation où cette méthode conviendrait mal.
4. Quelle preuve vous convaincrait qu'une implémentation fonctionne ?

# Chapitre 4 Révision

## Consignes de restitution

Enseignez les idées centrales de listes chaînées à quelqu'un qui n'a jamais programmé. Utilisez les analogies du chapitre, mais nommez aussi l'information exacte mémorisée par suivre la flèche suivante et opposez-l'à sentinelles. Terminez en expliquant comment chaque processus sait quand s'arrêter.

## Carte des connexions

| Concept | Se relie à | Demandez-vous |
|---|---|---|
| Suivre la flèche suivante | une insertion peu coûteuse quand l'emplacement est déjà connu | Que change quand l'entrée grandit ? |
| Simplement ou doublement chaînée | choisir la flexibilité de navigation | Que change quand l'entrée grandit ? |
| Pourquoi l'accès aléatoire est lent | reconnaître le compromis entre l'accès et la modification | Que change quand l'entrée grandit ? |
| Sentinelles | simplifier un code autrement fragile | Que change quand l'entrée grandit ? |

## À essayer sur papier

Choisissez cinq objets du quotidien qui correspondent à l'idée de sentinelles. Dessinez la représentation, effectuez une opération à la main, et annotez seulement l'information qui a changé. Expliquez ensuite pourquoi l'information inchangée satisfait toujours la promesse centrale du chapitre.