# Structures De Données et Types de Données Abstraits

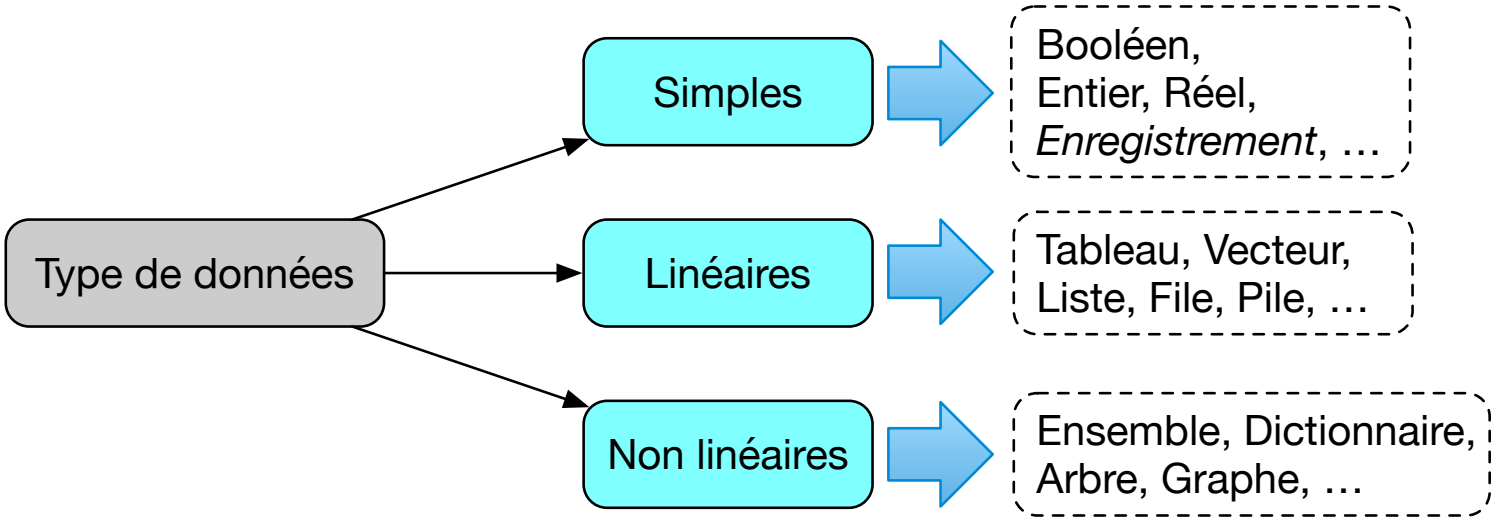
**Sébastien Jean**

IUT de Valence
Département Informatique

v1.0, 6 janvier 2026

IUT Valence
Université Grenoble Alpes

# Type de données




- **Identification d'une information** par un ensemble de valeurs (domaine) et d'opérations
  - **Simples** : une donnée est une valeur unique (*atomique*)
  - **Linéaires** : une donnée est une *collection* de valeurs, organisée en séquence
  - **Non linéaires** : une donnée est une *collection* de valeurs, organisée sans lien séquentiel

IUT Valence
Université Grenoble Alpes

# Le cas particulier des enregistrements

- Un **enregistrement** est un type de données qui peut être considéré comme simple ou non linéraire
- Il est **composé de plusieurs valeurs** appelées champs ou membres
  - Le nombre de champs est **fixe**
  - Les champs sont **nommés** et peuvent être de **type différent**
  - Les opérations se limitent à la **lecture et l'affectation des champs**
  - Accès aux champs via la **notation pointée** (`variable.champs`)
- Exemples :
  - Une date composée de 3 valeurs entières (jour, mois, année)
  - Un point composé de 2 valeurs réelles (abscisse, ordonnée)

IUT Valence
Université Grenoble Alpes

# Enregistrement et pseudo-code : exemple de Date

```
ENREGISTREMENT Date

   CHAMPS jour    : Entier
   CHAMPS mois    : Entier
   CHAMPS annee   : Entier

FIN ENREGISTREMENT
```

```
FONCTION est_valide (d : Date) : booléen

   SI d.jour < 0 OU d.mois < 0 ALORS
      RETOURNER FAUX

   ...

FIN FONCTION
```

# Type de données linéaires et non linéaires usuels

- N.B. : Lorsque l'on parle de collections, on fait une distinction entre :
  - **Statique** : le nombre de valeurs est fixe
  - **Dynamique** : le nombre de valeurs évolue
- Chaîne de caractères
  - Sequence ordonnée de caractères (dynamique)
- Tableau, Vecteur
  - Collection de valeurs repérées par leur position, statique (tableau) ou dynamique (vecteur)

IUT Valence
Université Grenoble Alpes

# Type de données linéaires et non linéaires usuels (suite)

- Liste, File, Pile
  - Groupe de valeurs organisées par une relation prédécesseur/successeur, avec notion de début et fin
  - Politique d'accès
    - Ajout/retrait n'importe où pour une **liste**
    - Ajout au début, retrait au début pour une **pile** (*LIFO : Last In First Out*)
    - Ajout en fin, retrait au début pour une **file** (*FIFO : First In First Out*)

IUT Valence
Université Grenoble Alpes

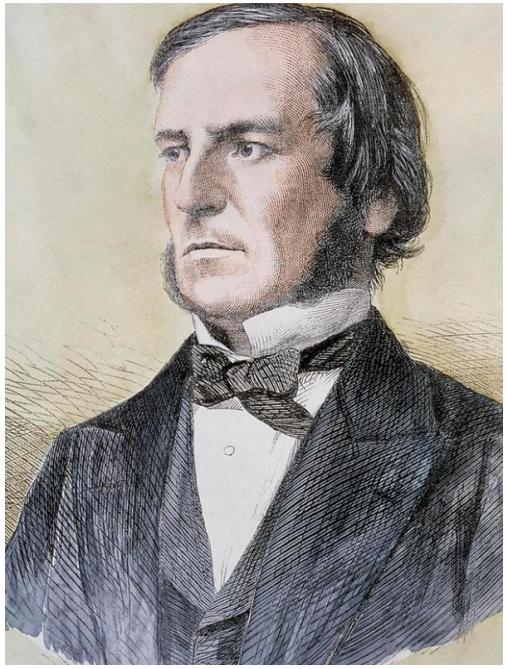
# Type de données linéaires et non linéaires usuels (suite)

- Dictionnaire
  - Groupe de paires clé/valeur, où les clés sont uniques
- Ensemble
  - Groupe de valeurs sans duplicata, sans notion de position
- Arbre
  - Groupe de noeuds où chaque noeud contient une valeur et est associé à d'autres noeuds par une relation parent/enfant
- Graphe
  - Groupe de noeuds où chaque noeud contient une valeur et est associé à d'autres noeuds par une relation dirigée et valuée

IUT Valence
Université Grenoble Alpes

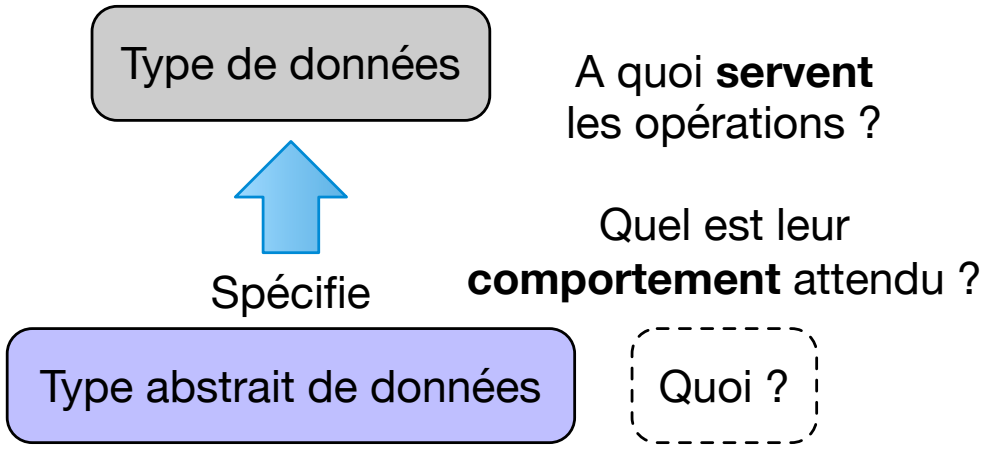
# Type de données : Booléen

- **Domaine** :
  - {VRAI, FAUX}
- **Opérations** :
  - ET, OU, NON (cf. algèbre de Boole)



IUT Valence
Université Grenoble Alpes

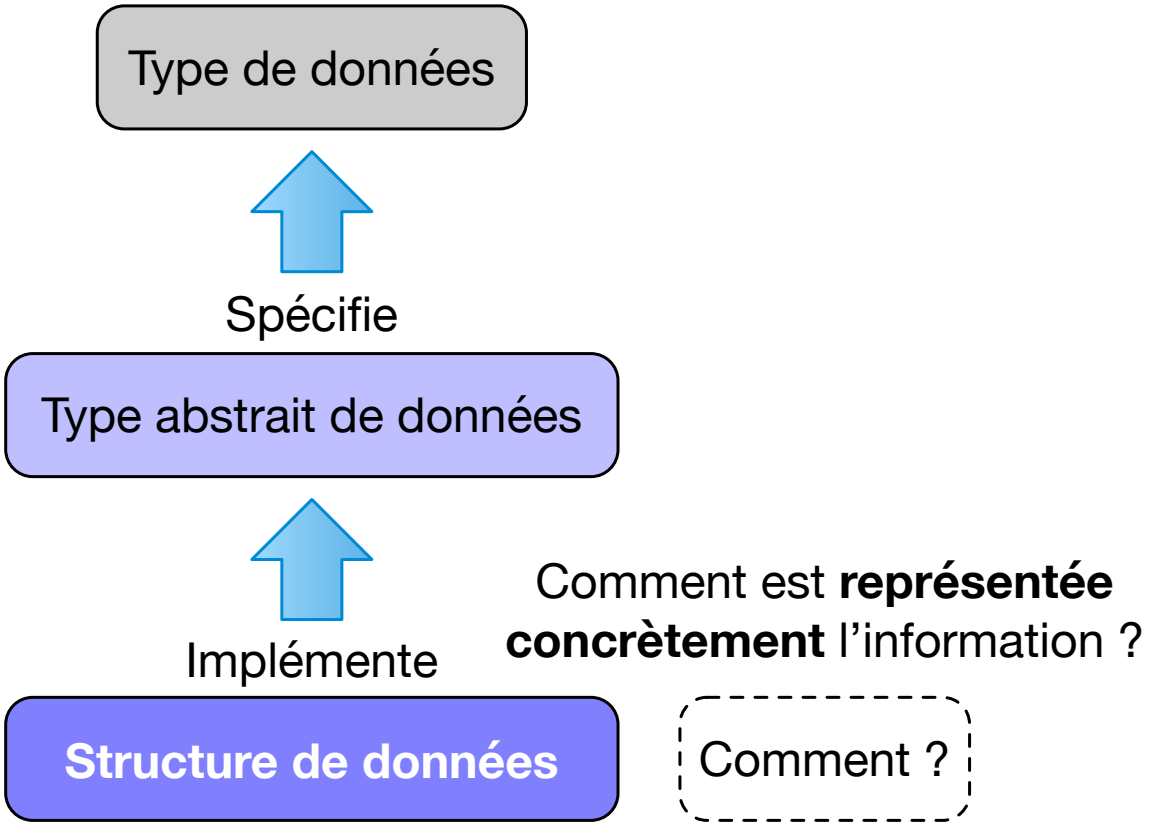
# Types de Données Abstrait (TDA)




- **Spécification mathématique** d'un type de données
  - **Nom**
  - **Dépendances**
  - **Opérations**
  - **Pré-conditions**
  - **Axiomes**

IUT Valence Université Grenoble Alpes

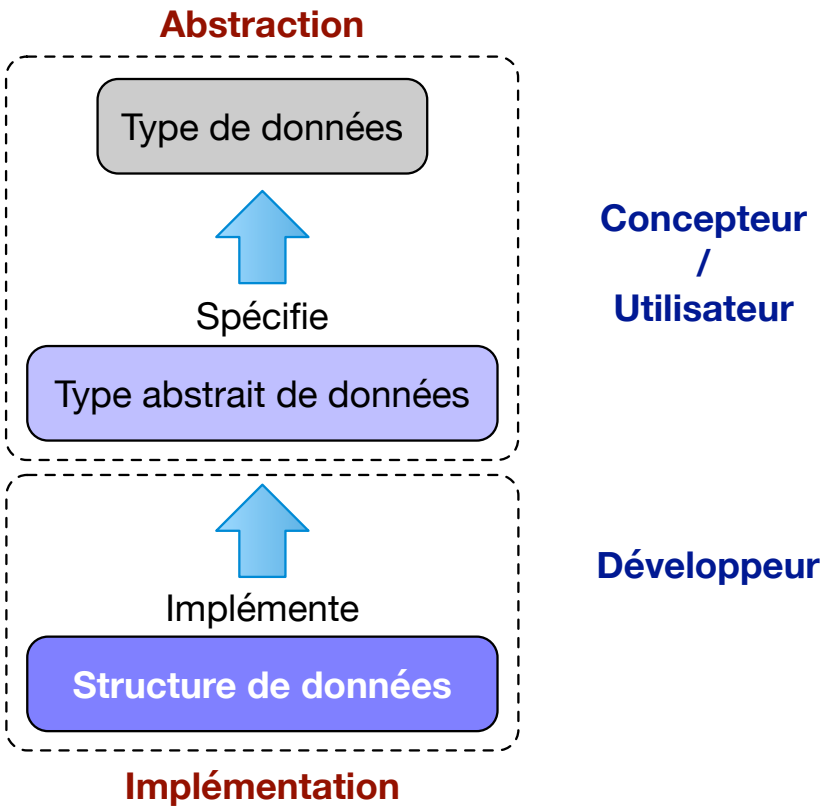
# Structure De Données (SDD)




- Implémentation concrète (langage / architecture) d'un TDA
  - N.B. : certains TDA simples (ex. booléens, entiers, flottants) se projettent sur des *types machine*

IUT Valence Université Grenoble Alpes

# Barrière de l'abstraction




- Les TDA masquent la réalité de l'implémentation aux **utilisateurs**
- Pour un même TDA, plusieurs SDD peuvent être définies
  - Complexité différente pour les opérations par ex.
  - Algorithmes écrits **indépendamment** d'une implémentation

IUT Valence Université Grenoble Alpes

# TDA : opérations

- 3 catégories :
  - **Constructeurs** : créent une donnée parmi les possibles
  - **Transformateurs**
    - **Producteurs** : produisent une nouvelle donnée à partir d'une ancienne
    - **Mutateurs** : modifient une donnée
  - **Observateurs** : donnent des informations sur la donnée
- **Syntaxe** de la description d'une opération
  - nom : type operande1 $x$ ... $x$ type operandeN $\rightarrow$ type résultat

IUT Valence Université Grenoble Alpes

# TDA : Booléen

- **Nom** : Booléen
- **Dépendances** : *(aucune)*
- **Opérations** :
  - **Constructeurs** :
    - vrai : $\rightarrow$ Booléen
    - faux : $\rightarrow$ Booléen
  - **Transformateurs** :
    - non : Booléen $\rightarrow$ Booléen
    - et : Booléen $x$ Booléen $\rightarrow$ Booléen
    - ou : Booléen $x$ Booléen $\rightarrow$ Booléen
- *A suivre . . .*

IUT Valence
Université Grenoble Alpes

# TDA : pré-conditions et axiomes

- **Pré-conditions**
  - Conditions devant être satisfaites par les arguments des opérations
- **Axiomes**
  - Propositions logiques vraies décrivant le comportement d'une opération

IUT Valence
Université Grenoble Alpes

# TDA : Booléen (suite)

- **Pré-conditions** : *(aucune)*
- **Axiomes** :
  - $non(vrai()) = faux()$
  - $non(faux()) = vrai()$
  - $et(a, b) = et(b, a)$
  - $ou(a, b) = ou(b, a)$
  - $et(vrai(), a) = a$
  - $et(faux(), a) = faux()$
  - $ou(vrai(), a) = vrai()$
  - $ou(a, faux()) = a$

IUT Valence Université Grenoble Alpes

# TDA et types machines

- Dans notre ***pseudo-code***, les booléens, les entiers, les flottants et les caractères sont considérés comme des types préexistants et "primitifs"
  - On a fait l'hypothèse (parce que c'est presque toujours le cas) qu'ils se projettent sur des types "machines", et que les opérations deviennent des opérateurs
- Un booléen a une représentation immédiate en mémoire (sur très peu de place), l'opération `et` à 2 paramètres est remplacée par l'opérateur `ET` à 2 opérandes
- ...

IUT Valence Université Grenoble Alpes

# Type de données : `Chaîne de caractères`

- Type de données linéaire, dynamique
- **Domaine** :
  - Toutes les séquences de caractères affichables (chiffres, lettres, symboles)
- **Opérations** :
  - ajouter un caractère (affichable) en fin de chaîne
  - connaitre le nombre de caractères
  - retrouver un caractère à une position donnée
  - concaténer 2 chaînes
  - savoir si 2 chaînes sont égales
  - ...

IUT Valence Université Grenoble Alpes

# TDA : Chaîne de caractères

- **Nom** : Chaîne
- **Dépendances** : *Caractère*, *Entier*, *Booléen*
- **Opérations** :
  - **Constructeurs** :
    - creer : $\rightarrow$ Chaîne (vide)
  - **Transformateurs** :
    - ajouter : Caractère $x$ Chaîne $\rightarrow$ Chaîne
    - concatener : Chaîne $x$ Chaîne $\rightarrow$ Chaîne
  - **Observateurs** :
    - taille : Chaîne $\rightarrow$ Entier
    - egal : Chaîne $x$ Chaîne $\rightarrow$ Booléen
    - caractere : Entier $x$ Chaîne $\rightarrow$ Caractère
- *A suivre . . .*

IUT Valence Université Grenoble Alpes

# TDA : Chaîne (suite)

- **Pré-conditions** :
  - `caractere(i, c)` $\rightarrow$ `1` $\leq$ `i` $\leq$ `taille(c)`
- **Axiomes** :
  - $taille(creer()) = 0$
  - $taille(ajouter(k, s) = taille(s) + 1$
  - $taille(concatener(s1, s2)) = taille(s1) + taille(s2)$
  - $caractere(1, ajouter(k, creer()) = k$
  - $caractere(i, ajouter(k, s)) = k$ si $taille(s) = i - 1$
  - $caractere(i, ajouter(k, s) = caractere(i, s)$ si $taille(s) > i - 1$
  - $egal(s1, s2) = faux()$ si $taille(s1) \neq taille(s2)$
  - $egal(creer(), creer()) = vrai()$
  - $egal(ajouter(k, s1), ajouter(k, s2)) = egal(s1, s2)$

IUT Valence
Université Grenoble Alpes

# TDA : `Chaîne` (fin)

- Certaines opérations sont **dépendantes de l'implémentation**
  - Elles nécessitent de connaitre la structure de données pour être écrites
- D'autres opérations sont **indépendantes de l'implémentation**
  - Elles peuvent être écrites sans connaitre l'implémentation, mais simplement en s'appuyant sur les opérations dépendantes de l'implémentation

- Quid des opérations du type `Chaîne` ?

IUT Valence
Université Grenoble Alpes

# Fin !

IUT Valence
Université Grenoble Alpes