Ministère de l'enseignement supérieur et de la recherche scientifique
Université Djillali BOUNAAMA - Khemis Miliana (UDBKM)
Faculté des Sciences et de la Technologie
Département de Mathématiques et d'Informatique

Chapitre 4

# Programmation Fonctionnelle

**MI-GLSD-M1-UEM213 : Paradigmes de langages de Programmation**

**Noureddine AZZOUZA**

n.azzouza@univ-dbkm.dz

# Plan du Cours

1. Introduction et Historique
2. Importance et domaine d’utilisation
3. Concepts principaux
4. Langage LISP

# Introduction & Historique

# Introduction

Le **langage fonctionnel** appartient à la famille des langages déclaratifs.

- ✓ Pas de commandes, pas de séquencement.
- ✓ Basés sur la notion de FONCTION.
- ✓ L'accent est mis sur le concept d'**expression** et leur **évaluation**.
  - ❑ **Objets** primitifs : constantes, caractères….etc.
  - ❑ **Expressions** primitives : fonctions élémentaires.
  - ❑ **Enchaînement** des expressions, composition des fonctions.

# Paradigme Fonctionnelle

- ✓ Le modèle théorique de la programmation fonctionnelle est le **Lambda calcul** (λ-calcul) «Une théorie introduite par **CHURCH (1941)** ».
- ✓ Le programme est une fonction ou un groupe de fonctions.
- ✓ Les fonctions sont reliées par les relations suivantes :
  - ❑ Une fonction peut appeler une autre fonction.
  - ❑ Le résultat d'une fonction peut être argument d'une autre fonction
- ✓ Le calcul (programmation) est réalisé entièrement au moyen d'évaluation d'expression.

# Exemple

## Langage impérative

```
Fonction factoriel (n : integer) : integer ;
  Var : f : integer ;
  Begin
     F := 1 ;
     WHILE n>0 do
        Begin
           F: = f * n;
           N: = n -1;
        End;
     Factorial: = f;
  End;
```

## Langage Fonctionnel

```
FUN "fonction" factoriel (n)
If n>0 then
N * factoriel (n-1)
Else 1;
```

# Importance et points Forts

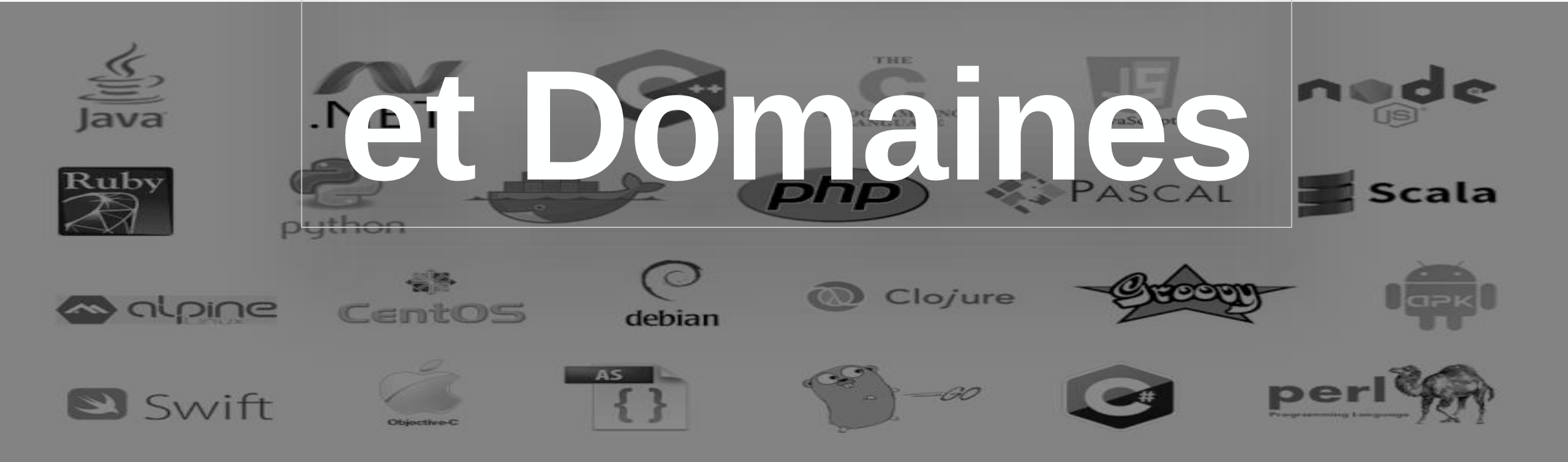
- ✓ Pas de notion d'emplacement mémoire (variable) [affectation-modification].
- ✓ Pas de commande et pas de structure de contrôle.
- ✓ Uniquement des valeurs qui résultent des fonctions et peuvent être utilisée par d'autres fonctions.
- ✓ Les langages fonctionnels ne sont pas contaminés par les effets de Bord, le séquencement explicite est d'autres caractéristiques vues comme repoussantes.

# Historique



- ✓ **1941 LAMBDA calcul**: est inventée par CHURCH pour étudier la calculabilité.
- ✓ **1959 LISt Processing**: premier langage fonctionnel symbolique pou l'IA conçu par JOHN M$^{C}$CARTHY.
- ✓ **1975 SCHEME**: un successeur plus simple et plus conformé que LISP
- ✓ **1987 ML** : premier langage à introduire le typage statique des fonctions polymorphes.
- ✓ **1990 HASKELL** : langage fonctionnel pur

# Importance et Domaines



# Importance et points Forts

- ✓ Les programmes peuvent être écrits :
  - ❑ Rapidement (surtout pour le prototypage)
  - ❑ De manière concise (utilisation d'une notation qui ressemble celle des mathématiques « simple »).
- ✓ Capacité de mener des raisonnements sur les programmes (raisonnement équationnel)
- ✓ Bien adapté au parallélisme (les arguments d'une fonction peuvent être évalués en parallèle).
- ✓ Ramener l'itération à la récursivité.

# Domaines d'utilisation

- ✓ Les langages fonctionnels sont utilisés dans de nombreuses applications : Emacs, GIMP, AUTOCAD, JEUX……
- ✓ **Domaines d'utilisation** :
  - ❑ Manipulation des symboles. « Compilation, traduction, intelligence artificielle, documents, web……»
  - ❑ Conception de logiciels complexes.
  - ❑ Apprentissage de la programmation.
- ✓ Elle est fortement présente dans les laboratoires et une faible pénétration dans le milieu industriel.

# Concepts Principaux

# Typage

- ✓ Le typage peut être statique ou dynamique.
  - ❑ **Statique** : lorsque les types sont vérifiés avant l'exécution.
  - ❑ **Dynamique** :les types sont vérifiés pendant l'exécution.
- ✓ La typage est soit faible soit fort :
  - ❑ **Fort** : toutes les erreurs de types sont détectées avant l'exécution « le programme ne donne jamais une erreur de type pendant son exécution.
  - ❑ **Faible** : pas toutes les erreurs de type sont détectées avant l'exécution.

# Inférence de type

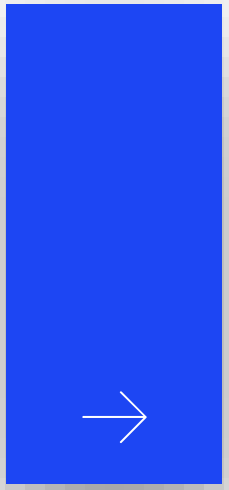
La capacité de dériver les types sans avoir besoin de signatures **explicites** pour les types.

- ✓ **Exemple** : **FUN** min ( a: real, b) = **if** a > b **then** b **else** a
  - ❖ Le type de a est donné (real), celui de b et de min sont **impliqués** automatiquement comme étant (real)

# Typage Polymorphique

typage d'une fonction **polymorphique** « une fonction qui peut être appelée avec des **arguments** de **différents** types ».

- ✓ **Exemple** : **HASKELL** Lengh :: [a] →Int
- ✓ Une fonction qui traite une liste d'objets de n'importe quel type a et renvoie un entier.



# Fonctions d’ordre supérieur

- ✓ les langages classiques traitent les fonctions comme des fonctions du premier ordre
- ✓ Les fonctions d’ordre supérieur sont traitées comme des valeurs de première classe
- ✓ Elles peuvent être passées en arguments et retournées comme résultats.
- ✓ Elles permettent un mécanisme d’abstraction sur des  valeurs
- ✓ Elles peuvent être stockées sous forme de structures de données ;
- ✓ Elles augmentent la réutilisation de code

# Évaluation Paresseuse

- ✓ La stratégie d'évaluation dans laquelle les arguments de fonctions sont évalués avant que la fonction soit appliquée s'appelle : **Eager évaluation** (évaluation stricte)
- ✓ **Exemple** : SCHEME (défaut)
- ✓ **L'évaluation paresseuse** est une stratégie d'évaluation des arguments de fonctions dans laquelle les arguments sont évalués uniquement quand ils sont nécessaires pour le calcul.

# Portée

- ✓ La portée des variables peut être statique ou dynamique :
- ✓ **Statique** : si les variables sont liées dans le contexte ou elles sont déclarées
  - ❑ **Exemple** : (scheme, ml, haskell).
- ✓ **Dynamique** : si les variables sont liées dans le contexte ou elles sont évaluées
  - ❑ **Exemple** : (LISP)

# Pattern matching

- ✓ C’est un concept très utilisé dans les langages fonctionnels modernes dont l’idée est l’utilisation du raisonnement équationnel dans la conception et la construction des programmes

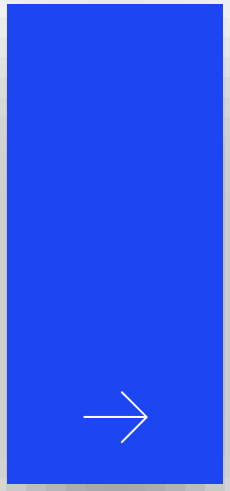
- ✓ Définir une même fonction par le biais de plusieurs fonctions et une seule de ces équations pourra être appliquée dans un contrôle.

# LISP

# Objets

- ✓ **LISP** : LISt Processor (1960)
- ✓ Les objets peuvent être
  - ❑ **atomes** ( symbole, nombre)
  - ❑ **liste** '( ' < suite d'objets >')'





# Structure d'un programme

- ✓ Toute donnée et tout programme Lisp est une S-expression (liste) engendrée par la grammaire suivante : **<Liste> --> <atome> ! ( <Liste> , <Liste>)**
- ✓ Les programmes LISP sont des listes particulières où le premier terme représente une fonction. les termes suivants sont les paramètres d'appel de cette fonction.
- ✓ **Exemple**: (Sin x) c'est sin(x)
- ✓ Lisp utilise donc la **notation préfixée** : (+ 3 2 ) c'est 3 + 2.

# Conditionnelle

(**IF** cond liste1 liste2 ..listen)

- ✓ Évalue suivant la valeur de Cond soit liste1 soit en "séquentielle«  liste2, liste3, ..., listen.
- ✓ La valeur retournée est celle de la dernière forme évaluée.
- ✓ On se limite a l'utilisation de (If Cond liste1 liste2).
- ✓ Valeurs de vérité : **T** pour **vrai**

  **Nil** et **0** pour **faux**
- ✓ T et Nil sont des atomes spéciaux.

# QUOTE

- ✓ QUOTE permet de distinguer les atomes spéciaux des constantes caractères dans les S-expressions
- ✓ (**Quote** objet) c'est retourner l'objet sans évaluation

**Exemple** :

- ❑ (Quote ( + 1 2) ) c'est équivalent à (+ 1 2)

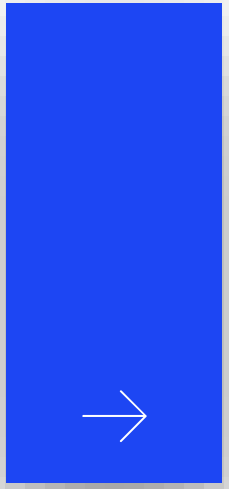
# Car

- ✓ (**Car** ListeNonVide): fournit la première composante de la liste ListeNonVide.
- ✓ **Exemple** :
  - ❑ (Car '(1 2 3) ) retourne 1.
  - ❑ ( Car '( (a b) 2 (3 c) ) ) retourne (a b).

# Cdr

- ✓ (**Cdr** ListeNonVide): fournit ListeNonVide privée de son premier élément.
- ✓ **Exemple** :
  - ❑ (Cdr '(1 2 3) ) retourne (2 3).
  - ❑ (Cdr '( (a b) 2 ( 3 c))) retourne (2 (3 c) )
  - ❑ (Cdr '(1) ) retourne 0.





# Cons

- ✓ (**Car** objet1 Liste): retourne une liste dont le premier terme est son premier argument (objet1) et les termes suivants sont ceux du second argument.
- ✓ **Exemple** :
  - ❑ (Const '1 '(2 3 4)) retourne (1 2 3 4 )
  - ❑ (Const '(1 2) '(3 4 5) ) ( (1 2) 3 4 5 )
- ✓ **Propriétés**:
  - ❑ (Car ( Cons x y ) ) = x
  - ❑ (Cdr ( Cons x y ) ) = y

# Primitives booléenne

- ✓ (**Atom** objet): retourne T si objet est un atome, Nil autrement.
- ✓ (**NumberP** obj) : retourne T ssi obj est un nombre, Nil autrement.
- ✓ (**SymbolP** Obj) :retourne T ssi obj est un symbole, Nil autrement.
- ✓ (**Consp** Obj) :retourne T ssi Obj est une liste non vide, Nil autrement.
- ✓ (**eq** Symb1 Symb2): teste l'égalité de 2 atomes.
- ✓ (**Equal** Obj1 Obj2): égalité de 2 objets.
- ✓ (**Null** Obj) : retourne T ssi Obj est une liste vide, Nil autrement.

# Définition de nouvelles fonctions

- ✓ ( **DE** NomDeFonction ( Var1 Var2 ..Varn) Liste1 Liste2 … Listen)
- ✓ Retourne NomDeFonction comme valeur.
- ✓ **Exemple 1** : Longueur d'une liste.

```
(DE Long(L)
   ( IF (Null L) 0
      (+ (Long ( Cdr L)) 1 )
   )
)
```

# Définition de nouvelles fonctions

✓ **Exemple 2** : Somme des éléments d'une liste.

```
(DE Som1 (L)
   ( IF (Consp L)
      (+ (Car L) (Som1 Cdr L))
      0
   )
)
```

```
(DE Som2(L)
   (IF (Consp L)
      (+ (Som2 (Car L)) (Som2 (Cdr L)) )
      (IF (NumberP L) L
      0
      )
   )
)
```

# Définition de nouvelles fonctions

✓ **Exemple 3** : Appartenance d'un atome à une liste.

```
(DE Exist (a L)
   (IF (Consp L )
      (Or (Exist a (Car L)) (Exist a (Cdr L)))
      (Eq a L)
   )
)
```

# Définition de nouvelles fonctions

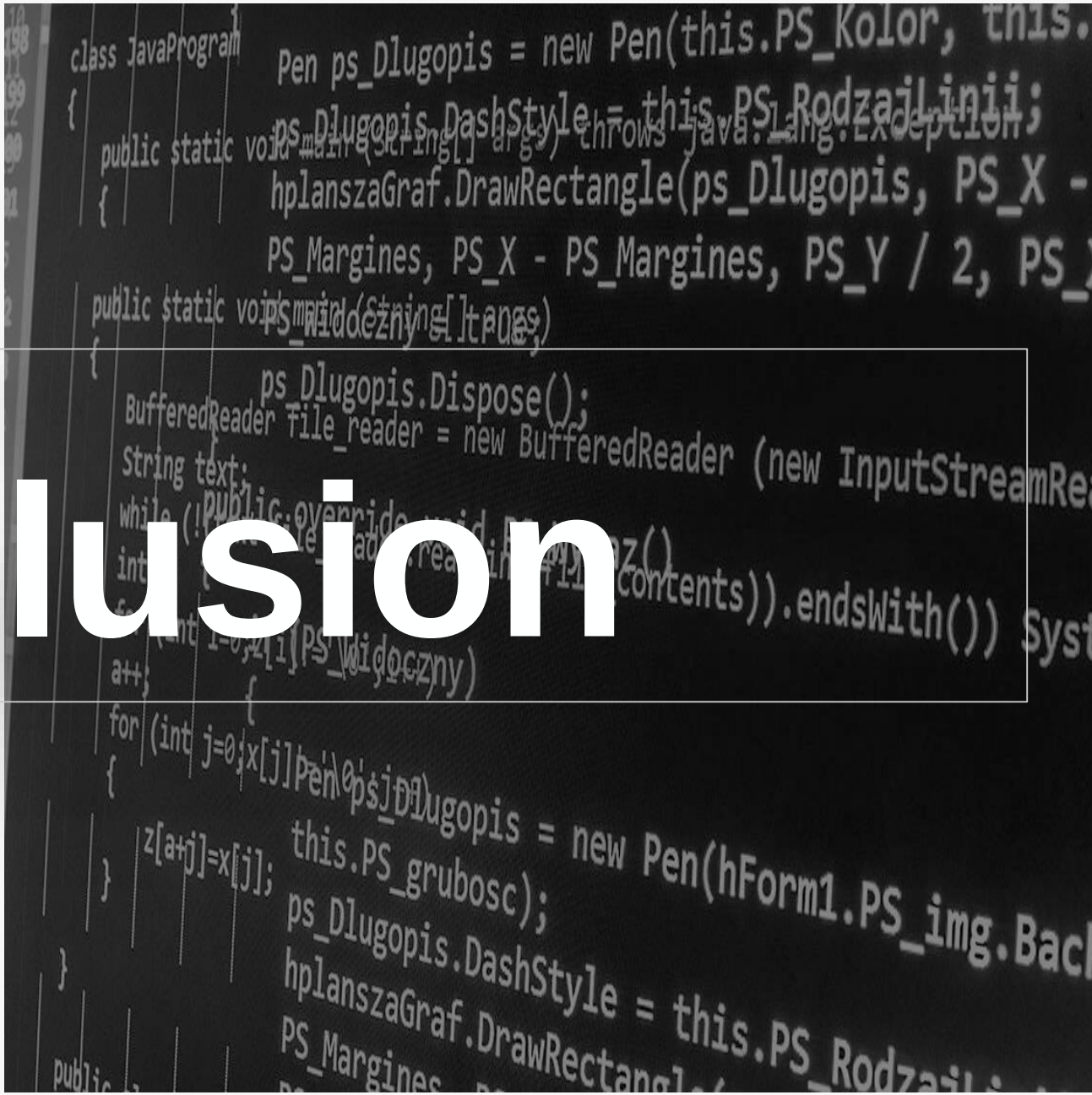
✓ **Exemple 4** : Inverser une liste d'atomes.

```
(DE Inverse(L)
   (Cons
   (Dernier L)
   (Inverse(Supprimer(Dernier L) L)
   )
)
```

```
(DE Dernier(L)
  (IF (Consp L)
     (IF (Null Cdr(L) )
     (Car L)
     (Dernier (Cdr L)))
  )
)
```

```
(DE Supprimer(a L)
   (IF (Consp L)
      (IF (Eq a Car(L))
       Cdr(L)
       Cons( Car(L) Supprimer(a Cdr(L)) )
   )
)
```

# Conclusion



# Conclusion

- ✓ Impact de la programmation fonctionnelle :
  - ❑ Analyse facile des programmes (démontrer qu'ils font les calculs désirés)
  - ❑ Transformation des programmes faciles (améliorer les performances).
  - ❑ L'ordre d'exécution a beaucoup moins d'importance qu'en programmation impérative.
  - ❑ Modèle de calcul plus proche des mathématiques ⟹ même techniques de preuve et de raisonnement (preuves par induction).