# Structures de contrôle

- Les **structures de contrôle** définissent *l'ordre de l'exécution* des instructions d'un programme
  - Cette ordre peut devenir non linéaire
- La plupart sont trouvées dans tous les langages
- Structures de **répétition** (boucles)
  - pour
  - tant que
  - jusqu'à ce que
- Structures de **condition** (alternatives)
  - si / sinon
  - selon / cas

Structures de blocs

- Instructions de saut
  - saut *
  - sous-programme *
- Instructions de terminaison
  - fin du passage
  - sortie de la boucle
  - sortie de la fonction/procédure
  - arrêt du programme
- Structures de gestion d'exceptions
  - signalement (lance)
  - interception (essaye / attrape)
  - réparation, reprise *

* *inexistantes en Matlab*

# Structure de répétition incrémentale « pour » (for)

- Décrit une **boucle incrémentale**
- **for** *variable* = *valeurs*
  *instructions*
  endfor
- Cette boucle est répétée tant de fois qu'il y a de *valeurs*
- Pour chaque *valeur* il y aura un nouveau **passage** de la boucle :
  - la *variable* sera affectée avec une valeur consécutive comme défini par *valeurs*
    - *variable* appelée **compteur** car c'est avec elle qu'on compte les passages
  - après, les *instructions* seront exécutées pour cette valeur de la *variable*
- Définition de *valeurs*
  - Matlab : *valeurs* doit être juste un vecteur quelconque contenant les valeurs consécutives avec lesquelles la *variable* doit être affectée
  - Autres langages : normalement plus pareil au pseudo-code (séquence numérique strictement croissante de 1 – d'où « boucle incrémentale »)
    - pseudo-code : pour a ← 1 à 4
    - Basic : For a = 1 To 4 Matlab : for a = [1:4]

# Répétition incrémentale – exemple

- Calculer la somme des éléments d'un vecteur numérique
- Développement de l'algorithme et du code du programme
  - **Indentation** : l'ensemble des instructions répétées sont décalées vers la droite par rapport à la ligne de début et à la ligne de fin de la boucle
  - **Indentation** : les actions qui sont exécutées en séquence (la deuxième toujours suit la première) sont mises en ligne, précisément l'une sous l'autre

Algorithme en pseudo-code

```
fonction somme_vect(v)
  entrées : v – vecteur de nombres
  sorties : somme – nombre
  intermédiaires : taille_v – nombre
        no_elem – nombre
  taille_v ← taille(v)
  somme ← 0
  pour no_elem ← 1 à taille_v faire
        somme ← somme + v[no_elem]
  fin pour
  retourner somme
fin fonction
```

codification

Code en langage Matlab

```
function somme = somme_vect(v)

  taille_v = numel(v);
  somme = 0;
  for no_elem = [1:taille_v]
        somme = somme + v(no_elem);
  endfor

endfunction
```

En langage Matlab, on ne déclare pas les variables internes ni les types de variables

# Répétition incrémentale – analyse pas à pas

- Entrée exemplaire
  - somme_vect( [5 10 7 12] )
- Cours de l'exécution
  - somme = 0;
  - début de la boucle
    for no_elem = [1:**4**]
    [**1** 2 **3** 4]
  - 1er passage de la boucle
    no_elem = **1**
    (on ne le trouve pas dans le code mais c'est exécuté automatiquement au début de chaque passage)
    somme = somme + v(**1**)
    0 5
    alors somme ← 5

```
somme = 0;
for no_elem = [1:numel(v)]
      somme = somme + v(no_elem);
endfor
```

- 2e passage
  no_elem = 2
  somme = somme + v(2)
  5 10
  alors somme ← 15
- 3e passage
  no_elem = **3**
  somme = somme + v(**3**)
  15 7
  alors somme ← 22
- 4e passage
  no_elem = 4
  somme = somme + v(4)
  22 12
  alors somme ← 34
- il n'y a plus de *valeurs* pour *no_elem* alors la boucle est terminée
- la variable *somme* garde le résultat : 34 = 5 + 10 + 7 + 12

# Opérateurs de relation

- Les **opérateurs de relation** sont :
  1) égal : == (c'est différent de l'opérateur de l'affectation qui est = ! c'est quand même pas universel : il y a des langages où = désigne la relation)
  2) inégal : != ou ~= ou <> (compatibilité avec de différents autres langages)
  3) inférieur à : <
  4) inférieur ou égal à : <=
  5) supérieur à : >
  6) supérieur ou égal à : >=
  - Liste exhaustive et universelle
- Le résultat d'une expression relationnelle est une valeur logique
  - Représentation de valeurs logiques en Matlab :
    - en entrée : on peut utiliser des variables spéciales ou des nombres
    - vrai ↔ true ↔ 1  faux ↔ false ↔ 0
    - en affichage : toujours les nombres
- En Matlab, pour les matrices, le résultat sera une matrice de résultats élément par élément

# Opérateurs logiques

- Les **opérateurs logiques** servent à former des conditions composées

| | standard (C, Matlab...) | binaire (C) ou él./él. (Matlab) | certains autres langages (stand. et bin.) |
|---|---|---|---|
| ◆ et | && | & | And |
| ◆ ou | \|\| | \| | Or |
| ◆ non pas | ! | ~ | Not |

- Traitement d'opérandes
  - standard : chaque opérande est convertie en une valeur logique simple
  - élément par élément : l'opération est effectuée séparément sur chaque élément d'opérandes matricielles
  - binaire : chaque opérande est considérée comme nombre binaire et l'opération s'effectue sur chaque des ces « 0 » ou « 1 » séparément
    - seulement utilisé par des professionnels
  - certains langages ne les discernent pas ou les discernent selon le contexte
- Les opérateurs de relation et logiques (standard) sont indispensables pour la plupart des structures de contrôle

# Expressions logiques (booléennes)

- a==4
  *a égal 4*
  - > a=4;
    > a==4
    ans = 1 *vrai*
  - > a==8
    ans = 0 *faux*
- a==3 || b<5
  *a égal 1* ***ou*** *b inférieur à 5*
  - > a=4; b=-2;
    > a==3 || b<5 ⟺ false || true
    ans = 1 *vrai*
- a==3 && b<5
  *a égal 3* ***et*** *b inférieur à 5*
  - > a==3 && b<5 ⟺ false && true
    ans = 0 *faux*

- Logique booléenne
  - conjonction ( && ∧ · )
    - « *p* **et** *q* » est vrai seulement si *p* est vrai et *q* est vrai

| *p* \ *q* | F | V |
|---|---|---|
| F | F | F |
| V | F | V |

  - disjonction ( || ∨ + )
    - « *p* **ou** *q* » est vrai si *p* est vrai ou *q* est vrai (ce qui comprend aussi le cas où les deux sont vrais)

| *p* \ *q* | F | V |
|---|---|---|
| F | F | V |
| V | V | V |