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

codification

Code en langage Matlab

```
function somme = sommevecteur(v)

  tailleVec = numel(v);
  somme = 0;
  for numElem = [1:tailleVec]
        somme = somme + v(numElem);
  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
  - sommevecteur([5 10 7 12])
- Cours de l'exécution
  - somme = 0;
  - début de la boucle
    for numElem = [1:**4**]
    [**1** 2 **3** 4]
  - 1er passage de la boucle
    numElem = **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 numElem = [1:tailleVec]
      somme = somme + v(numElem);
endfor
```

- 2e passage
  numElem = 2
  somme = somme + v(2)
  5 10
  alors somme ← 15
- 3e passage
  numElem = **3**
  somme = somme + v(**3**)
  15 7
  alors somme ← 22
- 4e passage
  numElem = 4
  somme = somme + v(4)
  22 12
  alors somme ← 34
- il n'y a plus de *valeurs* pour *numElem* 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 des 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

Structure de condition « si / sinon » (if / else)

- Structure simple
  - **if** (*condition*)
    instructions
    endif
  - Si la *condition* est remplie, alors les *actions* sont exécutées
  - Sinon, aucune action n'est prise
- La *condition*
  - une **expression logique** quelconque, c.-à-d. une expression à laquelle peut être attribuée une valeur logique (vrai / faux)
- Structure avec alternative
  - if (*condition*)
    instructions
    else
    autres_instructions
    endif
  - Si la *condition* est remplie, alors les *instructions* sont exécutées
  - Sinon, les *autres_instructions* sont exécutées

Structure de condition « si / sinon » (suite)

- Structure avec alternatives multiples
  - if (*condition_1*)
    instructions_1
    elseif (*condition_2*)
    instructions_2
    elseif (*condition_3*)
    instructions_3
    …
    elseif (*condition_n*)
    instructions_n
    else
    instructions_nn
    endif
- La partie « else » n'est pas obligatoire
  - Il peut n'y avoir aucune alternative finale
  - si *condition_1* est remplie (*true* ou 1)
    - les *instructions_1* sont exécutées
    - aucune autre condition n'est vérifiée
  - sinon (si *condition_1* n'est pas remplie alors sa valeur est *false* ou 0)
    - la *condition_2* est vérifiée
    - si *condtion_2* est remplie
      - *instructions_2* sont exécutées
      - aucune autre condition n'est vérifiée
    - sinon (*condition_2* n'est pas remplie)
      - *condition_3* est vérifiée
      - …
  - si aucune des *conditions* 1 à *n* n'est remplie
    - *instructions_nn* sont exécutées

Structure « si / sinon » – exemples

- Structure avec alternative :
  - Mémoriser le supérieur de deux nombres dans une troisième variable
  - ```
    if (a > b)
        max = a;
    else
        max = b;
    endif
    ```
  - Ça marche aussi quand les deux nombres sont égaux (dans ce cas il n'est pas important la valeur de quelle variable est copiée parce que c'est la même valeur)
  - **Indentation** : l'ensemble des instructions qui sont exécutées sous une condition sont décalées vers la droite par rapport à la ligne où cette condition est définie
- Alternatives multiples :
  - Déterminer le signe d'un nombre (1 ou –1) ; 0 s'il est 0
  - ```
    if (n > 0)
        sgn = 1;
    elseif (n < 0)
        sgn = -1;
    else
        sgn = 0;
    endif
    ```
  - La condition « n == 0 » n'apparaît pas parce qu'elle est déduite des deux premières conditions (si aucune n'est remplie, alors *n* doit être égal 0)

Structures imbriquées

- Exemple :
  - Calculer la somme des éléments positifs seulement d'un vecteur
  - ```
    somme = 0;
    for numElem = [1:numel(v)]
        if v(numElem) > 0
            somme = ...
            somme+v(numElem);
        endif
    endfor
    ```
  - p. ex.
    ```
    v = [1 -2 4 -8 16 -32]
    ```
- La structure niveau plus bas (*if*) doit être fermée (*endif*) avant que la structure niveau plus haut (*for*) soit fermée (*endfor*)

- Structures du même type :
  - Calculer la somme des éléments d'une matrice
  - ```
    somme = 0;
    for iLigne = [1:rows(M)]
        for iColonne = [1:columns(M)]
            somme = ...
            somme+M(iLigne,iColonne);
        endfor
    endfor
    ```
  - p. ex.
    ```
    M = [1 -2; 4 -8; 16 -32]
    ```
- Il sera assumé que la première structure fermée est la dernière ouverte, donc le premier *endfor* correspond au *for iColonne* et non pas au *for iLigne*

Structure de condition « selon / cas » (switch / case)

- Cas générique :
  - **switch** (*variable*)
    case *valeur_1*
    instructions_1
    case *valeur_2*
    instructions_2
    …
    case *valeur_n*
    instructions_n
    otherwise
    instructions_nn
    endswitch
- Remplacement pour « if » afin de ne pas répéter « elseif »
- Il doit y avoir au moins un *case*
- La partie *otherwise* n'est pas obligatoire

- Attention avec d'autres langages basés sur le C
  - À la fin d'un cas (*case*) il faut mettre *break*
  - Sinon, l'exécution continuera avec les *instructions* du cas suivant
    - Ce n'est pas désiré dans la plupart des cas (pas tous les cas quand même)
- Cas combinés
  - case {194 233}
    banque="Credit Agricole"
  - est équivalent à :
    case 194
    banque="Credit Agricole"
    case 233
    banque="Credit Agricole"

Structure de condition « selon / cas » – exemple

- À comparer :
  - Afficher le descriptif d'une note d'un employé exprimée comme nombre
  - ```
    switch (note)
        case 5
            disp("excellent");
        case 4
            disp("bien");
        case 3
            disp("passable");
        case 2
            disp("insuffisant");
        otherwise
            disp("nombre incorrect");
    endswitch
    ```
  - Plus lisible et plus court
  - Plus facile de changer du nom de la variable ou ajouter plus d'options
- ...avec une solution basée sur *if* (le résultat est identique) :
  - ```
    if (note == 5)
        disp("excellent");
    elseif (note == 4)
        disp("bien");
    elseif (note == 3)
        disp("passable");
    elseif (note == 2)
        disp("insuffisant");
    else
        disp("nombre incorrect");
    endif
    ```
- Afin de pouvoir utiliser *switch*, il est nécessaire que chaque condition :
  - soit basée sur l'égalité (==)
  - concerne la même variable (ici, *note*)