



Année 2010-2011

# Cours de programmation

CM1  
Programmation impérative

Ruding LOU  
[ruding.lou@ensam.fr](mailto:ruding.lou@ensam.fr)

# **Plan du cours**

## **1. Algorithme et programmation informatique**

Algorithme informatique  
Pseudo code  
Langage de programmation

## **2. Variable**

Type

## **3. Expression**

Affectation  
Opération

## **4. Appel d'un sous-programme**

Lire  
Ecrire

## **5. La structure de contrôle**

Test (branchement conditionnel)  
Boucle

## **6. Type de donnée avancé**

Tableau  
Structure de donnée

## **7. Programme principal**

# Algorithme

Un **algorithme** est

- un processus systématique de résolution, par le calcul, d'un problème permettant de présenter les étapes vers le résultat à une autre personne physique (un autre humain) ou virtuelle (un calculateur);
- une méthode effective ou suite d'opérations permettant de donner la réponse à un problème;
  - Séquentiel : les opérations s'exécutent en séquence
  - Parallèle: les opérations s'exécutent sur plusieurs processeurs en parallèle
  - Réparti (distribué): les opérations s'exécutent sur un réseau de processeurs

# Algorithme

Dans la vie quotidienne

- **Exécution** des algorithmes
  - On ouvre un livre de recettes de cuisine.
  - On se sert d'un mode d'emploi.
- **Fabrication et faire exécuter** quelqu'un des algorithmes
  - On indique un chemin à un touriste égaré.
  - On fait chercher un objet à quelqu'un par téléphone.

# Algorithme

- Un **algorithme**, c'est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.
  - Si l'algorithme est **juste**, le résultat est le résultat voulu, et le touriste se retrouve là où il voulait aller.
  - Si l'algorithme est **faux**, le résultat est, disons, aléatoire.
- Pour fonctionner, un **algorithme** doit contenir uniquement des instructions compréhensibles par celui qui devra les exécuter
  - (c'est d'ailleurs l'un des points délicats pour les rédacteurs de modes d'emploi).

# Algorithme informatique

- Les ordinateurs sont capables de comprendre principalement quatre catégories d'ordres (instructions). Ces quatre familles d'instructions sont :
  - les affectations (assignation) de variables
  - les opérations (la lecture / écriture, les opérations arithmétiques)
  - les tests / structures (branchement) conditionnelles
  - les boucles
- Un algorithme informatique se ramène donc toujours au bout du compte à la combinaison de ces quatre petites briques de base.
- La taille d'un algorithme ne conditionne pas en soi sa complexité : de longs algorithmes peuvent être finalement assez simples, et de petits très compliqués.

# Programmation

- **Programme**
  - Instructions pouvant être réalisées par un ordinateur
- **Paradigmes :**
  - impératif, procédural, objet, fonctionnel, évènementiel, déclaratif, logique, etc.
- **Langage de programmation**
  - Langage pouvant être analysé (in)directement par une machine
    - FORTRAN(1954) : Impérative, calculs
    - C (1972) Impérative, procédurale
    - C++ (1983), JAVA (1995): Orientée objet
- **Compilation**
  - code source -> code exécutable (syntaxe + sémantique)

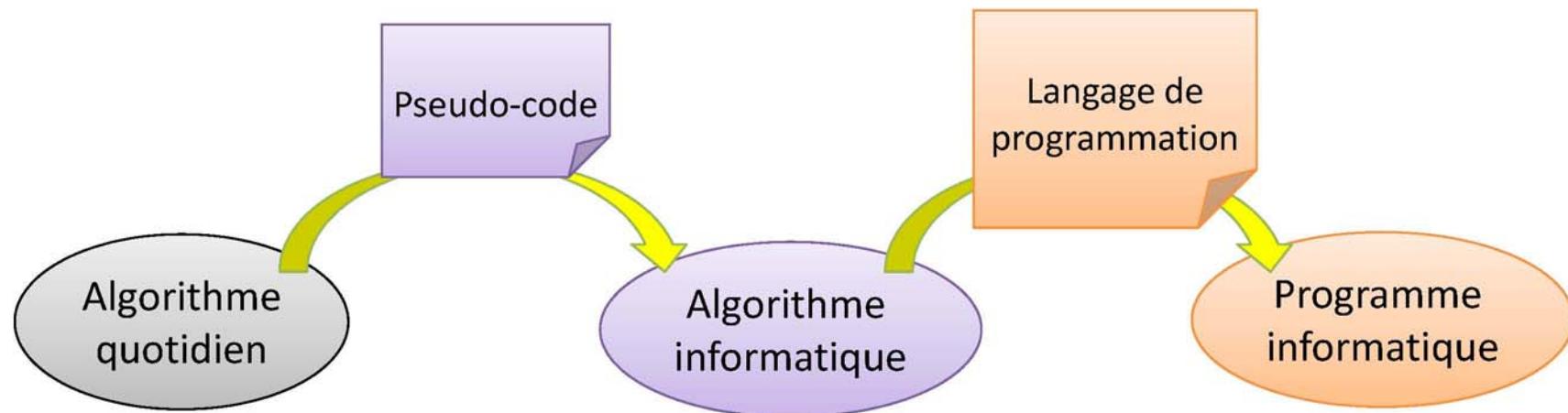
# Langage VB

- Aux *Arts et Métiers* le langage retenu est le VB
- Avantages
  - Syntaxe verbeuse initialement développée pour l'apprentissage de la programmation
  - Bien intégré aux outils et environnements Microsoft (Windows)
  - IDE (Visual Studio (payant), Express (gratuit)), permet de créer visuellement des interfaces graphiques Windows
- Inconvénients
  - Confusion des paradigmes (impérative, procédurale, orienté objet)
  - Performances, gestion de la mémoire
  - Quelques spécificités discutables (types de données, passages par référence, syntaxe verbeuse et confuse)
- **Quand on connaît un paradigme, on peut très facilement changer de langage : seule la syntaxe et quelques spécificités changent**

# Pseudo-code

- Le **pseudo-code** est une façon de décrire un algorithme via un langage qui est l'intermédiaire entre une langue parlée et un code informatique.
- L'écriture du **pseudo-code** à l'aide de mots appartenant à un langage parlé a normalement lieu avant l'écriture du code informatique lui-même.
- L'écriture en **pseudo-code** permet souvent de bien prendre toute la mesure de la difficulté de la mise en œuvre de l'algorithme, et de développer une démarche structurée dans la construction de celui-ci.
- L'algorithme écrit en **pseudo-code** peut ensuite être traduit dans un langage de programmation quelconque. Un pseudo-code bien écrit pourrait être utilisé par deux programmeurs différents qui le traduirraient dans deux langages différents mais avec le même résultat.
- Exemple de pseudo-code:  
**Variable a en Entier**  $\leftarrow 99$   
**Variable b en Entier**  $\leftarrow a + 1$   
**Ecrire b**

# Pseudo-code ↔ langage de programmation



Recette de cuisine

Itinéraire

Mode d'emploi

**Variable a en Entier** ← 99  
**Variable b en Entier** ← a+1  
**Ecrire b**

--- Algo. en Français

**Dim a As Integer** = 99  
**Dim b As Integer** = a + 1  
**Console.WriteLine(b)**

--- Prog. en VB

# Variable

- Pourquoi ?
  - Besoin de stocker provisoirement des valeurs
  - Info de provenance divers : nombre, texte, matrice, image ...
- Qu'est ce que c'est ?
  - Espace mémoire (adresse, taille)
  - Accessible par un identificateur
- Comment utiliser : la déclaration + initialisation d'une variable
  - Déclarer l'identificateur d'une variable (=> adresse dans le mémoire)
  - Préciser le type (=> taille de l'espace mémoire)
  - Initialiser la variable (=> stocker une valeur)
  - Utiliser et modifier la valeur de la variable en désignant son identificateur

Début du }  
prog.

Corps du prog.

# Variable

- Syntaxe:

**Dim identificateur As Type = valeur**

- Identificateur

- lettres, chiffres, '\_' (Attention ! mots réservés et signification)

- Type: taille espace mémoire, valeurs, sémantique

- Booléens ( $2^1$ ): **Boolean**

- numérique
- Octet ( $2^8$ ): **Byte**, Entier simple ( $2^{16}$ ): **Integer**, Entier long ( $2^{32}$ ): **Long**
    - Réel simple ( $2^{32}$ ): **Single**, Réel double ( $2^{64}$ ): **Double**

- alphanumérique
- Caractère: **Char**
    - Chaînes de caractères : **String**
    - Etc.

**Variable a en Entier**

**Variable a\_1 en Réel**

**Variable nom en Chaîne de caractères**



**Dim a As Integer**  
**Dim a\_1 As Single**  
**Dim nom As String**

# Variable

- Type
  - Booléen ( $2^1$  bits): **TRUE** ou **FALSE**
  - Entier: chiffres signé ( $2^8$ ,  $2^{16}$ ,  $2^{32}$  bits):  $-2^7$  à  $2^7 - 1$        $-2^{15}$  à  $2^{15} - 1$
  - Réel ( $2^{32}$ ,  $2^{64}$  bits): chiffres signés avec décimales
  - Alphanumérique : symboles de code **ASCII**

# Constant

- Une constante est une expression littérale
  - 1, 2.34E-56
  - "A"
  - "bonjour"
  - True
- Une constante est une variable spéciale qui ne change pas de valeur

**Variable age en Entier**

**Constante pi en Réel**

**Constante ensam en Chaîne de caractères**

Dim a As Integer

Const pi As Single

Const ensam As String

# Expression: affectation

- Attribuer une valeur littérale à une variable

- Syntaxe: **var = exp**
- Sémantique: la valeur de **exp** est évaluée et devient, après conversion au type de **var**, la nouvelle valeur de **var**

Variable	a	en Entier	← 99
Variable	b	en Réel	
Variable	nom	en Chaîne de caractères	← "toto"
Constante	ensam	en Chaîne de caractères	← "Ecole Nationale Supérieure d'Arts et Métiers"
Constante	pi	en Réel	← 3.1415926

b ← 9.9  
nom ← "titi "

Obligation d'initialiser une **constante**

Un caractère et une chaîne de caractères sont toujours notés entre les guillemets

```
Dim a As Integer = 99
Dim b As Single
Dim nom As String = "toto"
Const ensam As String = "Ecole Nationale Supérieure d'Arts et Métiers"
Const pi As Single = 3.1415926
```

b = 9.9
nom = "titi"

# Expression: affectation

- Attribuer une valeur littérale à une variable

- Syntaxe: **var = exp**
- Sémantique: la valeur de **exp** est évaluée et devient, après conversion au type de **var**, la nouvelle valeur de **var**

```
Variable nom en Chaîne de caractères ← "toto"
Constante pi en Réel           ← 3.15
Constante pi_txt en Chaîne de caractères ← "3.1415926"
Variable msg en Chaîne de caractères ← "pi est 3.15"
```

```
pi ← 3.1415926
msg ← "pi est 3.1415926"
```

Obligation d'initialiser  
une **constante**

Un caractère et une chaîne de caractères  
sont toujours notés entre les guillemets

```
Dim nom As String = "toto"
Const pi As Single = 3.15
Const pi_txt As String = "3.1415926"
Dim msg As String = "pi est 3.15"
```

```
pi = 3.1415926
msg = "pi est 3.1415926"
```

# Expression: affectation

- Attribuer une valeur stockée dans un variable à une autre variable

```
Variable pi_num en Réel  
Constante pi en Réel           ← 3.15  
Constante pi_txt en Chaîne de caractères ← "3.1415926"  
Variable msg en Chaîne de caractères  
  
pi_num ← pi  
msg ← "pi_txt"  
msg ← pi_txt
```

```
Dim pi_num As Single  
Const pi As Single = 3.15  
Const pi_txt As String = "3.1415926"  
Dim msg As String
```

```
pi_num = pi  
msg = "pi_txt"  
msg = pi_txt
```

msg vaut pi\_txt

msg vaut 3.1425926

# Expression: opération

- Opérateurs arithmétiques

- Addition : **+**
- Soustraction : **-**
- Multiplication : **\***
- Division : **/**
- Modulo (reste division entière) : **Mod**
- Quotient entière : **\**
- Puissance: **^**

- Format :  $exp_1 \text{ OP } exp_2$

- Le type du résultat est le type nécessitant le plus grand nombre de bits parmi celles des deux expressions concernées.

```

Dim a As Single = 0
Dim b As Single = 0

b = a + 1      b: 1
b = b - a      b: 1
b = a * b      b: 0
b = 100 / 8    b: 12.5
b = b Mod 5    b: 2
b = 11 \ 3     b: 3
b = b ^ b      b: 27

```

# Expression: opération

- Opérateurs alphanumérique
  - Concaténation de chaînes deux caractères : &

```
Dim a As String = "Arts"  
Dim m As String = "Métiers"  
Dim s As String = a &" et " &m
```

- Opérateurs logique

- Format:  $exp_1 \text{ OP } exp_2$

- ET (Conjonction) : And (AndAlso)
- OU (Disjonction) : Or (OrElse)
- OU eXclusif (Distinction): Xor

|s: Arts et Métiers

- Format: **OP**  $exp$

- NON (Négation): Not

- Type de paramètres

- Opérande (s): Booléenne
- Résultat: Booléenne

# Expression: opération

- Opérateurs logique
  - Table de vérité, Vrai:**1** Faux:**0**

a	b	a <b>ET</b> b	a <b>OU</b> b	A <b>OUeXclusif</b> b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

a	<b>NON</b> a
0	1
1	0

```
Dim a As Boolean = True  
Dim b As Boolean = False
```

```
b = a And b
```

```
b = a Or b
```

```
b = a Xor b
```

```
b = Not b
```

```
b = a Or True
```

```
b = True Xor True
```

```
b: False
```

```
b: True
```

```
b: False
```

```
b: True
```

```
b: True
```

```
b: False
```

# Expression: opération

- Opérateurs comparaison

- Format:  $exp_1 \text{ OP } exp_2$ 
  - Égalité:  $=$
  - Supériorité stricte:  $>$
  - Supériorité non stricte:  $\geq$
  - Infériorité stricte:  $<$
  - Infériorité non stricte:  $\leq$
  - Différence:  $\neq$

- Type de paramètres
  - Résultat: Booléenne

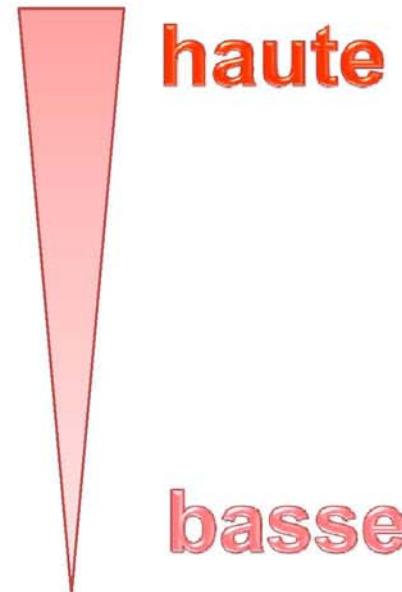
```
Dim a As Integer = 10  
Dim b As Single = 10.0  
Dim r As Boolean = False
```

$r = a > b$	r: False
$r = a \geq b$	r: True
$r = a \neq b$	r: False
$r = a < 11$	r: True
$r = a \leq 11$	r: True

# Expression: opération

- Priorités opérateurs

( )			
<b>Not</b>	<b>^</b>		
*	/	<b>Mod</b>	\
+	-		
<	<b>&lt;=</b>	>	<b>&gt;=</b>
=	<b>&lt;&gt;</b>		
<b>And</b>			
<b>Or</b>			
= (affectation)			



- L'évaluation d'une expression se fait par ordre décroissant de priorités.

# Expression: opération

- Abréviations



$exp_1 \text{ OP } = exp_2$

$exp_1 = exp_1 \text{ OP } exp_2$

Dim a As Integer = 10	
Dim b As Integer = 20	
b = a + b	b: 30
b += a + b	b: 70
b *= a	b: 700
b *= a / 5	b: 1400

b += a + b $\Leftrightarrow$ b = b + a + b $\Leftrightarrow$ b = 30 + 10 + 30
b *= a $\Leftrightarrow$ b = b * a
b *= a / 5 $\Leftrightarrow$ b = b * a/5



# Appel d'un sous-programme

- Appel d'un sous-programme modifiant l'état courant
- Syntaxe:

**identificateur**(argument0, argument1, ... , argumentN)

- Types de sous-programmes

- Retourne une valeur (expression): **fonction**

```
somme = addition(a, b)  
racine = racineCarree(a)  
saisie = lireDepuisConsole()
```

- Ne retourne pas de valeur: **procédure**

```
ecrireDansConsole(a+b)  
ecrireDansConsole(addition(a, b))  
additionSansRetour(operande1, operande2, somme)  
nettoyerConsole()
```

```
ecrireDansConsole( additionSansRetour(op1, op2, somme) )
```



```
ecrireDansConsole()
```



# Entrées / Sorties : Ecrire

- Ecrire (afficher) sur la console
- Syntaxe (procédure avec un paramètre):

**Console.WriteLine** ( chaîne de caractères )

**Console.WriteLine** ( chaîne de caractères )

- La 2<sup>ème</sup> commande permet d'écrire la chaîne de caractères et de mettre une fin de ligne
- La chaîne de caractères :

- Concaténation

```
Console.WriteLine("bonjour" & " toto")
```

- Conversion

```
Console.WriteLine("il y a " & 23 & " personnes")
```

- Symboles VB permettant de changer une ligne: **vbCrLf** :

```
Console.WriteLine("toto : 2 ans" & vbCrLf &"titi : 3 ans")
```

**Variable a, b en Integer**

$a \leftarrow 10$

$b \leftarrow a^2$

Interaction avec utilisateurs  
- Connaître l'état des variables

**Variable a, b en Entier**

$a \leftarrow 10$

$b \leftarrow a^2$

**Ecrire "a^2 = " &b**

**Dim a, b As Integer**

$a = 10$

$b = a^2$

**Console.WriteLine("a^2 = " &b)**

# Entrées / Sorties : Lire

- Lire (saisir) une valeur depuis la console
- Syntaxe (fonction sans paramètre):

chaine de caractères = **Console.ReadLine ()**

**Variable a, b en Integer**  
a ← 10  
b ← a<sup>2</sup>

Interaction avec utilisateurs  
- Modifier l'état des variables

- Permet de saisir puis de retourner une chaîne de caractères jusqu'à la fin de ligne (retour chariot)
- La saisie de types multiples est converti en chaîne de caractère :

**Variable a en Entier**

**Variable s en Chaîne De Caractères**

**Ecrire "saisir un entier "**

**Lire a**

**Ecrire "saisir un message"**

**Lire s**

**Ecrire s &"a<sup>2</sup> = " & a<sup>2</sup>**

```
Dim a As Integer
Dim s As String
Console.WriteLine("saisir un entier")
a = Console.ReadLine()
Console.WriteLine("saisir un message")
s = Console.ReadLine()
Console.WriteLine(s &"a2 = " &a2)
```



# La structure de contrôle: Les tests

Exprimé sous forme de pseudo-code,  
la programmation de notre touriste donnerait donc quelque chose du genre :

*Allez tout droit jusqu'au prochain carrefour*

*Si la rue à droite est autorisée à la circulation Alors*

*Tournez à droite*

*Avancez*

*Prenez la deuxième à gauche*

*Sinon*

*Continuez jusqu'à la prochaine rue à droite*

*Prenez cette rue*

*Prenez la première à droite*

*Fin Si*

}

**Expression de condition**

}

**Blocs d'instructions**

# La structure de contrôle: Les tests

- Trois formes possibles pour un test

- Syntaxe 1:

```
If expression Then
    Instruction
End If
```

**Si** expression est vraie,  
**alors** instruction est exécutée,  
**sinon** on ne fait rien.

- Syntaxe 2:

```
If expression Then
    Instructions1
Else
    Instructions2
End If
```

**Si** expression est vraie,  
**alors** instruction1 est exécutée,  
**sinon** instruction2 est exécutée.

- Syntaxe 3:

```
If expression1 Then
    Instructions1
ElseIf expression2 Then
    Instructions2
( . . . )
ElseIf expressionN Then
    InstructionsN
Else
    InstructionsN+1
End If
```

**Si** expression1 est vraie,  
**alors** instruction1 est exécutée;  
**Sinon si** expression2 est vraie,  
**alors** instruction2 est exécutée;  
(...)  
**sinon si** expressionN est vraie,  
**alors** instructionN est exécutée  
**sinon** instructionN+1 est exécutée.

# La structure de contrôle: Les tests

- Trois formes possibles pour un test

– Syntaxe 1:

```
If expression Then
    Instruction
```

```
End If
```

– Syntaxe 2:

```
If expression Then
    Instructions1
```

```
Else
```

```
Instructions2
```

```
End If
```

– Syntaxe 3:

```
If expression1 Then
```

```
Instructions1
```

```
ElseIf expression2 Then
```

```
Instructions2
```

```
(...)
```

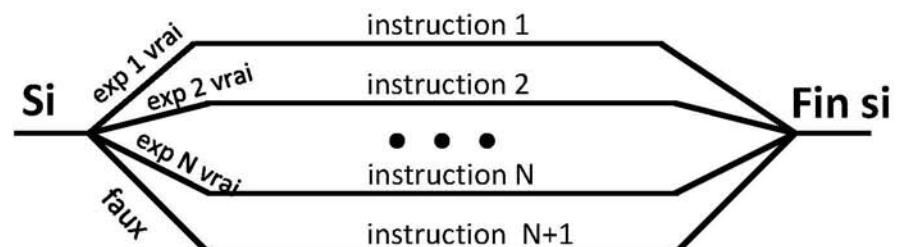
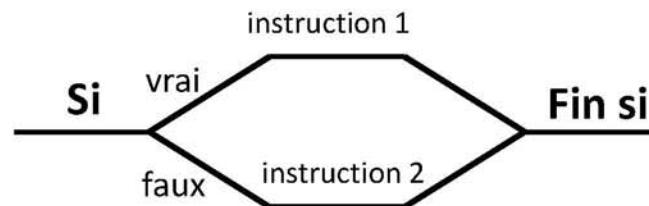
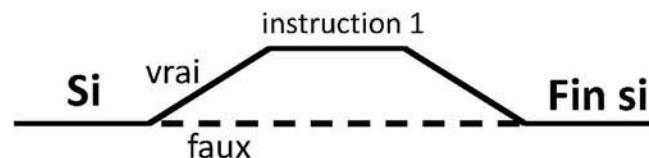
```
ElseIf expressionN Then
```

```
InstructionsN
```

```
Else
```

```
InstructionsN+1
```

```
End If
```



# La structure de contrôle: Les tests

- Expression de condition
  - Les opérateurs de comparaison : =, <>, >, <, >=, <=
- Expression composé
  - Prenons le cas « Toto est inclus entre 5 et 8 ». En fait cette phrase cache non une, mais deux conditions. Car elle revient à dire que
    - " Toto est supérieur à 5 "
    - et
    - " Toto est inférieur à 8 ".
  - Il y a donc bien là deux conditions, reliées par un opérateur logique, le mot **ET**.
- Quatre opérateurs logiques : ET, OU, NON, et XOR

```
If expression Then  
| Instruction  
End If
```

# La structure de contrôle: Les tests

**Variable a, b en Entier**

**Ecrire "saisir premier entier"**

**Lire a**

**Ecrire "saisir deuxième entier"**

**Lire b**

**Si a > b Alors**

**Ecrire "le premier est plus grand"**

**Sinon Si a = b Alors**

**Ecrire "les deux sont égaux"**

**Sinon**

**Ecrire "le deuxième est plus grand"**

**Fin Si**

```
Dim a, b As Integer
```

```
Console.WriteLine("saisir premier entier")
```

```
a = Console.ReadLine()
```

```
Console.WriteLine("saisir deuxième entier")
```

```
b = Console.ReadLine()
```

```
If a > b Then
```

```
    Console.WriteLine("le premier est plus grand")
```

```
ElseIf a = b Then
```

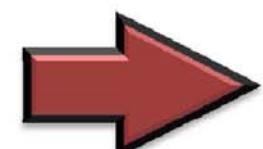
```
    Console.WriteLine("les deux sont égaux")
```

```
Else
```

```
    Console.WriteLine("le deuxième est plus grand")
```

```
End If
```

## Trois exécutions

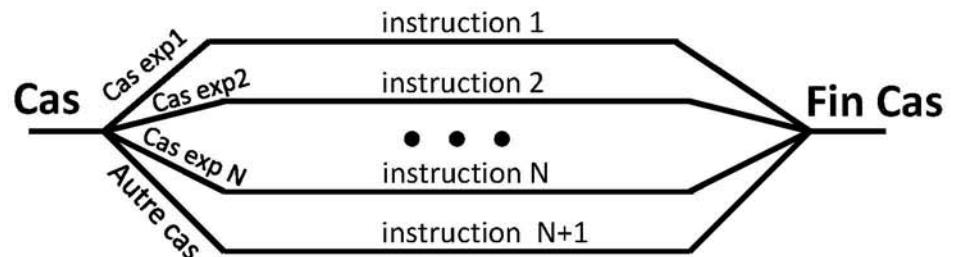


# La structure de contrôle: Les tests

**Objective:** Exécute un groupe d'instructions parmi plusieurs autres, selon la valeur d'une expression.

## Syntaxe:

```
Select Case variable
    Case exp1
        Instructions 1
    Case exp2
        Instructions 2
    ...
    Case expN
        Instructions N
    Case Else
        Instructions N+1
End Select
```



```
Dim opt As Integer
Console.WriteLine("Entrez votre choix")
opt = Console.ReadLine()
Select Case opt
    Case 0
        Console.WriteLine("Egal à 0")
    Case 1 To 5
        Console.WriteLine("Entre 1 et 5 ")
    Case 6, 7, 8
        Console.WriteLine("Entre 6 and 8")
    Case Is <= 10
        Console.WriteLine("Egal à 9 ou 10")
    Case Else
        Console.WriteLine("Hors de 0 à 10")
End Select
```

# La structure de contrôle: Les boucles

- Répéter une action **While** et **For**

- sous une condition

- Syntaxe 1:

**While** *expression*

*instruction*

**End While**

**Tant que** *expression*

*instruction*

**Fin Tantque**

- Syntaxe 2:

**Do While** *expression*

*instruction*

**Loop**

**Faire Tantque** *expression*

*instruction*

**Boucle**

- Syntaxe 3 (**exécuter une fois "instruction" avant le test**):

**Do**

*instruction*

**Faire**

*instruction*

**Boucle Tantque** *expression*

Ruding LOU

ruding.lou@ensam.fr

# La structure de contrôle: Les boucles

- Répéter une action **While** et **For**
  - sous une condition

**Variable a en Entier**  $\leftarrow 1$

**Tantque**  $a < 50$

$a \leftarrow a * 2$

**Fin Tantque**

**Ecrire** "a = " &a

```
Dim a As Integer = 1
```

```
While a < 50
```

```
    a = a * 2
```

```
End While
```

```
Console.WriteLine("a = " &a)
```

**Variable a en Entier**  $\leftarrow 1$

**Faire**

**Ecrire** "Saisir... positif et inférieur à 25 ... "

**Lire** a

**Boucle Tantque**  $a \leq 0$  **OU**  $a \geq 25$

**Ecrire** "RDV à " &a & " heures"

```
Dim a As Integer = 1
```

```
Do
```

Console.WriteLine("Saisir un entier positif  
et inférieur à 25")

```
a = Console.ReadLine()
```

```
Loop While a <= 0 Or a >= 25
```

```
Console.WriteLine("RDV à " &a & " heures")
```

# La structure de contrôle: Les boucles

- Répéter une action **While** et **For**
  - Sous un nombre d'itérations imposé
  - Syntaxe:

**For** *iterateur = initial To final Step increment*  
*instruction*  
**Next** *iterateur*

- Opération équivalente à

*iterateur = initial*  
**While** *iterateur <= final*  
*instruction*  
*iterateur = iterateur + increment*  
**End While**

**Pour** *iter ← init à fin Pas inc*  
*instruction*  
**iter Suivant**

*iter ← init*  
**Tantque** *iter <= final*  
*instruction*  
*iter ← iter + inc*  
**Fin Tantque**

# La structure de contrôle: Les boucles

- Répéter une action **While** et **For**
  - Sous un nombre d'itérations imposé

**Variable**  $a, b, c, i$  en Entier

**Ecrire** "saisir un entier"

**Lire**  $a$

**Faire**

**Ecrire** "saisir l'exposant positif"

**Lire**  $b$

**Boucle Tantque**  $b < 1$

$c \leftarrow a$

**Pour**  $i \leftarrow 1$  à  $b-1$  **Pas** 1

$c \leftarrow c * a$

**i Suivant**

**Ecrire**  $a$  &"^" & $b$  &"=" & $c$

```
Dim a, b, c, i As Integer
```

```
Console.WriteLine("Saisir un entier")
a = Console.ReadLine()
```

```
Do
```

```
    Console.WriteLine("Saisir l'exposant pos...")
```

```
    b = Console.ReadLine()
```

```
Loop While b < 1
```

```
c = a
```

```
For i = 1 To b - 1 Step 1
```

```
    c = c * a
```

```
Next i
```

**Exercice 4**



```
Console.WriteLine(a &"^" &b &"=" &c)
```

# Type de données avancés: Tableau

- Pourquoi:

- Besoin de créer plusieurs éléments du même type

```
Dim n1 As Integer
```

```
Dim n2 As Integer
```

...

```
Dim n100 As Integer
```

- Besoin de parcourir les éléments

```
n1 = Console.ReadLine()
```

```
n2 = Console.ReadLine()
```

...

```
n100 = Console.ReadLine()
```

- Solution: tableau

- Object structurés contenant plusieurs éléments du même type

# Type de données avancés: Tableau

- Syntaxe:

**Dim identificateur (taille) As Type**

- Exemples de déclaration sans/avec initialisation

Dim nombres (10) As Integer

Dim notes () As Single = {5.5, 6.6, 7, 10, 8, 2, 15, 20, 9, 10.1}

Dim etudiants () As String = {"toto", "titi", "tata"}

- Utilisation

- taille, indice sont de type entier

- indice est de **0 à taille – 1**

- Accès à chaque élément par **identificateur ( indice )**

- Exemples d'utilisation

notes (0) = 5.5

etudiants (2) = "marie"

Dans Visual Basic le nombre de cases allouées est en fait **taille+1**, ce qui est très différent que les autres langages de programmation

Pour redimensionner un tableau on peut utiliser la commande  
**Array.Resize(identificateur, nouvelleTaille)**

# Type de données avancés: Tableau

- Exemple

**Variable i en Entier**

**Variable moy en Réel**  $\leftarrow 0.0$

**Tableau notes (10) en Réel**

**Pour i  $\leftarrow 0$  à 9 Pas 1**

**Ecrire** "saisir la note (" &i+1 &")"

**Lire** notes(i)

**i Suivant**

**Pour i  $\leftarrow 0$  à 9 Pas 1**

    moy  $\leftarrow$  moy + notes(i)

**i Suivant**

moy = moy/10

**Ecrire** "la moyenne est " &moy

```
Dim i As Integer
Dim moy As Single = 0.0
Dim notes(10) As Single
```

```
For i = 0 To 9 Step 1
    Console.WriteLine("Saisir la note(" &i+1 &")")
    notes(i) = Console.ReadLine()
```

Next i

```
For i = 0 To 9 Step 1
    moy = moy + notes(i)
Next i
```

```
moy = moy / 10
Console.WriteLine("la moyenne est " &moy)
```

# Type de données avancés: Tableau

- Tableaux multi-dimensionnels
  - Chaque élément d'un tableau peut-être un tableau
- Syntaxe (tableau de dimension N)

**Dim identificateur (taille1, taille2, ..., tailleN) As Type**

**Variable i, j en Entier**

**Tableau matrice (10, 5) en Integer**

**Pour i ← 0 à 9 Pas 1**

**Pour j ← 0 à 4 Pas 1**

**matrice (i, j) ← i \*10+j**

**j Suivant**

**i Suivant**

**Pour i ← 0 à 9 Pas 1**

**Pour j ← 0 à 4 Pas 1**

**Ecrire " " &matrice(i, j)**

**j Suivant**

**Ecrire retour-chariot**

**i Suivant**



ISIT

ISIT

ISIT

Dim i, j As Integer

Dim matrice(10, 5) As Integer

For i = 0 To 9 Step 1

For j = 0 To 4 Step 1

matrice(i, j) = i\*10 + j

Next j

Next i

For i = 0 To 9 Step 1

For j = 0 To 4 Step 1

Console.WriteLine(matrice(i, j) & " ")

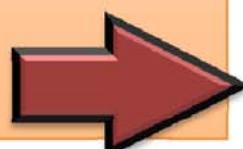
Next j

Console.WriteLine("")

Next i

**Exercice 5**

gramma  
tation in



# Type de données avancés: Structure

- On peut créer des nouveaux types de données composés
- Syntaxe de la création

**Public Structure Nouveautype**

**Public element1 As Type**

**Public element2 As Type**

    ...

**Public element2 As Type**

**End Structure**

- Syntaxe de l'utilisation

**Dim var As Nouveautype**

var . element

```
Public Structure Livre
    Public titre As String
    Public annee As Integer
    Public auteur As String
    Public enStock As Boolean
End Structure
```

```
Dim livre As Livre
livre.titre = "Introduction à VB"
livre.annee = 2010
livre.auteur = "Bill Gates"
livre.enStock = True
```

# Programmation impérative

- Exécution d'un programme commence par la procédure principale qui s'appelle **Main()**
- Déclaration d'une procédure **Sub ... End Sub** (*sub-routine*)
- Déclaration des variables se fait au tout début

**Procédure Main( )**

**Variable i en Entier**

**Variable s en Chaîne de caractères**

**EcrireLigne "Début du programme"**

**Ecrire "Saisir votre nom: "**

**Lire s**

**Pour i ← 0 à 4 Pas 1**

**EcrireLigne i &": " &s**

**i Suivant**

**Fin Procédure**

Sub Main()

Dim i As Integer

Dim s As String

Console.WriteLine("Début du programme")

Console.Write("Saisir votre nom: ")

s = Console.ReadLine()

For i = 0 To 4 Step 1

Console.WriteLine(i &": " &s)

Next i

End Sub