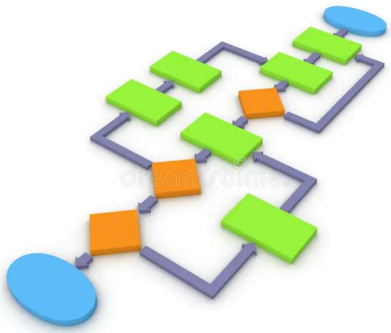
Applied Computer Tools



Programming in VBA-Introduction
(+ algorithmics)

frantz.fournier@univ-lorraine.fr

Variables and operators in VBA

Variables in programming language

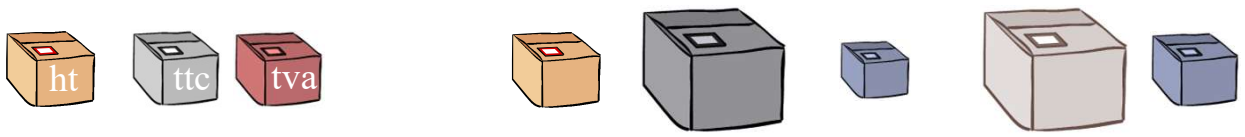
- In any language, data are stored in *variables* to be used for computation. e.g. :

```
ht = 10.0
tva = 0.2
ttc = ht * (1 + tva)
```

VBA reads the last statement from right to left.
It first computes the right-hand side formulae: ht * (1 + tva)
Then store the value in the left variable (ttc)
(consider the « = » sign as an affectation symbol (like ←):
ttc ← ht * (1 + tva)

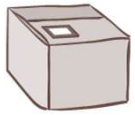
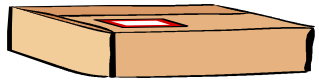
- But, before using any variables, you must first declare its **name** and its **type** (to prepare the name and the « size » of the memory box which is necessary to store its value).

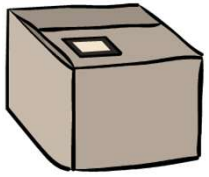
E.g.:



Variables types

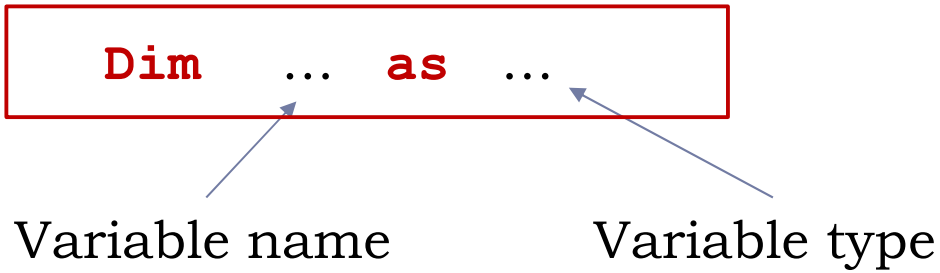
VBA propose the following type of variables:

- NUMERICAL
  - **Integer** (from -32768 to 32767) 
  - **Long** (from -2147483648 to 2147483647) 
  - **Single** (from -3.402823 10^{38} to 3.402823 10 ^{38}) 
  - **Double** (from -1:79769313486232 10^{308} to 1.79769313486232 10^{308}) 
- CHARACTER
  - **String** (using " " as delimiters of the character string) 
- BOOLEAN
  - **Boolean** (with only two possible values : True or False) 
- OTHER
  - **Variant** any kind of content
  - **Object** Windows Object (presented later)



Declaring variables

Option explicit



Declaration examples:

```
Dim i as Long

Dim Delta as Double
Dim a as Double, b as Double, c as Double,

Dim Prenom as String, Nom as String

Dim Status as Boolean
Dim FourreTout as Variant
```

Variables in VBA

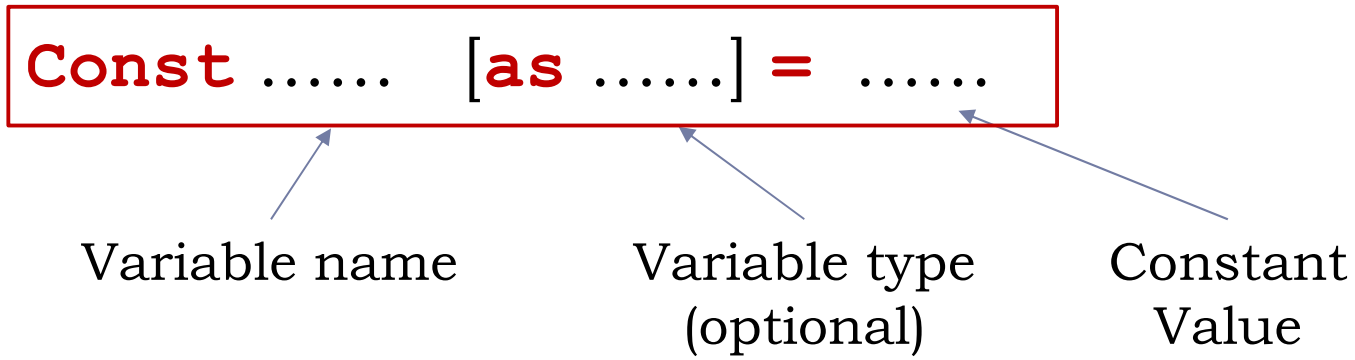
Example

```
Dim ttc as Double
Dim ht as Double, tva as Double
ht = 10.0
tva = 0.2
ttc = ht * (1.0+tva)
  '-------------------------------------------------------------------------
Dim Prenom as String, Nom as String
Dim Identite as String
Prenom = "Frantz"
Nom = "Fournier"
Identite = Prenom & " " & Nom
           ' & --> concatenation of strings
  '-------------------------------------------------------------------------
Dim Status as Boolean
Status = True                    ' or False
```

Note	Variable name do not contain “space”, operators,…
	Only simple letters, numbers or “_”

Declaring constant in VBA

Some of your variables will keep a constant value
Declare them as *constant*



Example:

```
Const Pi As Single = 3.14159265358979
```

or simply

```
Const Pi = 3.14159265358979
```

Operators

Classical operators

+ - * / and ^ for power (+ exponential 4e3 for 4×10^3)

Boolean operators are used to compare variables.

=	Equal
<>	Different
<	less than
>	Greater than
<=	Less or equal to
>=	Greater to equal to
Not	Opposite

- AND/OR to combine comparison

Examples

```
Dim x as Double
Dim y as Double
Dim z as Double
Dim Test as Boolean
x =  1.0
y = 3.0
Test = x > y
'-------------------
' The value of Test
' is False in this
' example
z = 22
Test = 20<z AND z<30
```

Using VBA functions

Calling integrated VBA functions

$$y = f(x)$$
$$z = f(x, y)$$

Examples:

```
x = exp(-2*t+1)          ' exponential
y = sqrt(27)             ' square root
z = log(x)               ' natural logarithm
zdB = 20*log10(z)        ' 10-based logarithm
p = abs(p)               ' absolute value
```

You can call VBA function using various syntaxes:

res = functionname val1, val2, val3

res = functionname(val1, val2, val3) *(prefer this syntax)*

res = functionname argname1:=val1 argname2:=val2 , argname3:=val3

where *functionname* is the function name
val1, val2, val3 are values or variables containing values
argname1, argname2,... are formal argument names in the function
res is the variable receiving the function result

Some usual VBA functions

- **Abs** : absolute value
- **Exp** : exponential
- **Log**: natural logarithm
- **Sqr**: square root

- **Sgn** : sign

- **Atn** : arctangent
- **Cos** : cosine
- **Sin** : sine
- **Tan** : tangent

- **Int**, **Fix** : nearest integer part
- **Rnd**: random number (single)

- **MsgBox**: message box
- **InputBox**: input box

- **Str**: convert Value to String
- **Val**: Convert String to Value

- **Loc**: locate a text in a string
- **Len**: length of a string
- **Lcase**/**Ucase**: convert to lower or upper case
- **Mid**: extract text from a string
- **InSt**: search position of a text in a string

- **IsEmpty**: test if argument is empty

- **Ubound**/**Lbound**: lower/upper index of a matrix

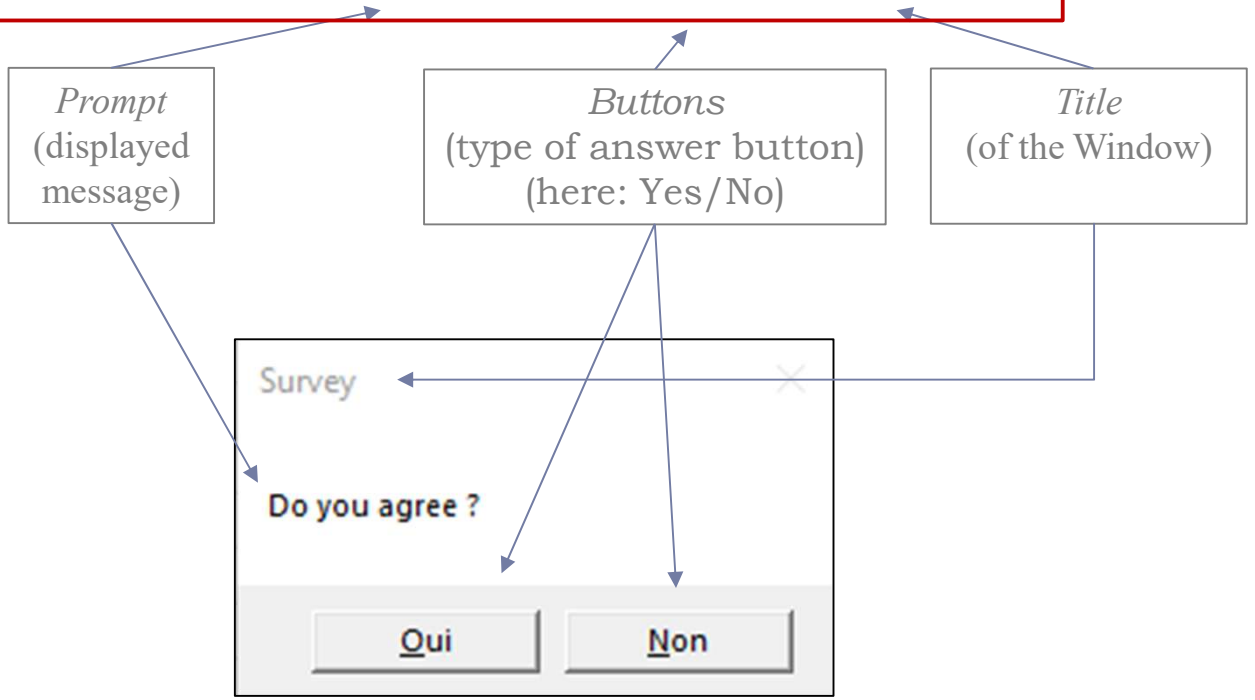
- **Date**: current date
- **Now**: current date and time

And many others…

Example with the *MsgBox* function

MsgBox function to display a message

```
Rep = MsgBox("Do you agree ?",4, "Survey")
```



Alternative

```
Rep = MsgBox (Buttons:=4,Prompt:="Do you agree ?", Title:="Survey")
```

Note that in the last example, no need to respect the argument order.

Coding MsgBox buttons

Constante	Valeur	Description
vbOKOnly	0	OK
vbOKCancel	1	OK Annuler
vbAbortRetryIgnore	2	Abandonner Recommencer Ignorer
vbYesNoCancel	3	Oui Non Annuler
vbYesNo	4	Oui Non
vbRetryCancel	5	Recommencer Annuler
vbCritical	16	
vbQuestion	32	
vbExclamation	48	
vbInformation	64	
vbDefaultButton1	0	Bouton par défaut : Bouton 1
vbDefaultButton2	256	Bouton par défaut : Bouton 2
vbDefaultButton3	512	Bouton par défaut : Bouton 3

Simply add the values to get the corresponding options in the MsgBox.

```
Rep = MsgBox ("Do you agree ?", _
                4+32, "Survey!")
```

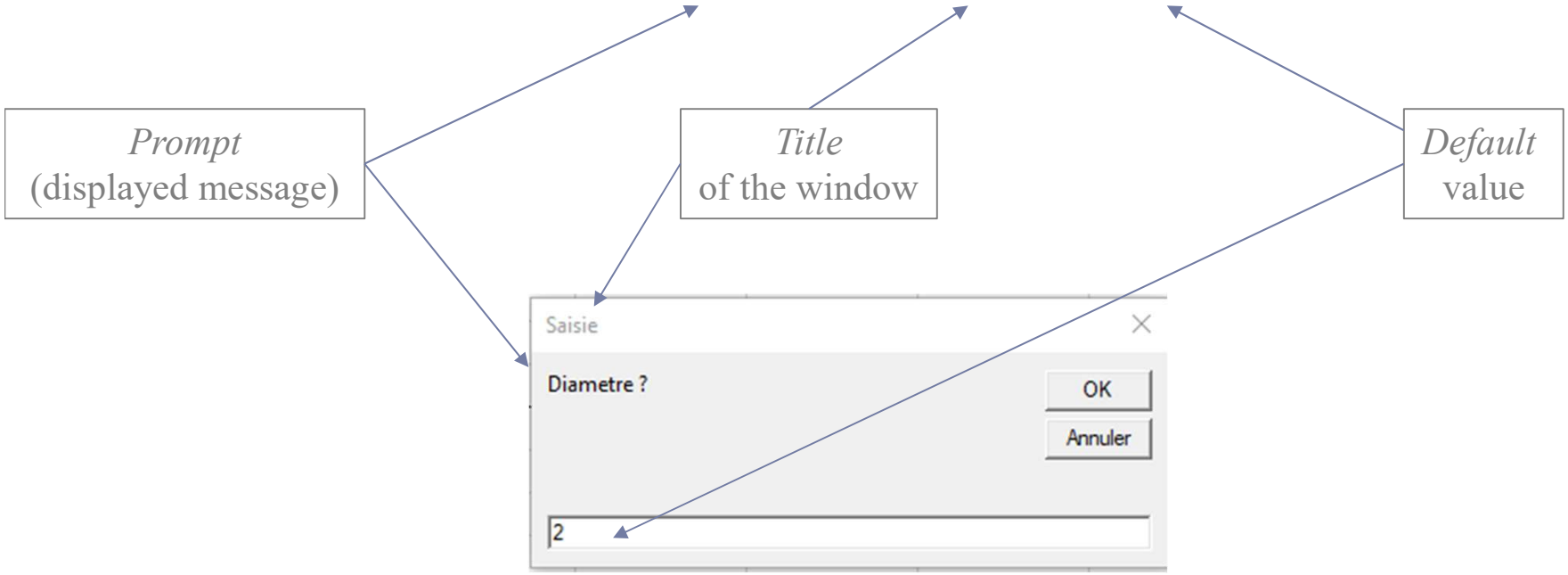
```
Rep = MsgBox ("Do you agree ?", _
              vbYesNo+vbQuestion, _
              "Survey")
```

Possible *Rep* values are : vbOK, vbYes, VbNo, VbCancel, vbAbort,…

Inputbox function

Ask a question to the user to get an additional information

```
Sub CircleArea()
Dim D as Double
' User value will be stored in variable D
D = InputBox("Diametre ?", "Saisie",2.0)
```




```
MsgBox(" Circle Area : " & string(3.14/4*D^2))
End Sub
```

Creating your own functions

Functions

You can create you own function in VBA to automate a series of usual operations.

Like VBA standard functions, your function will:

- Have a unique name
- Needs some information (arguments) to be processed
- Return one (unique) computed value

Creating your own function in VBA

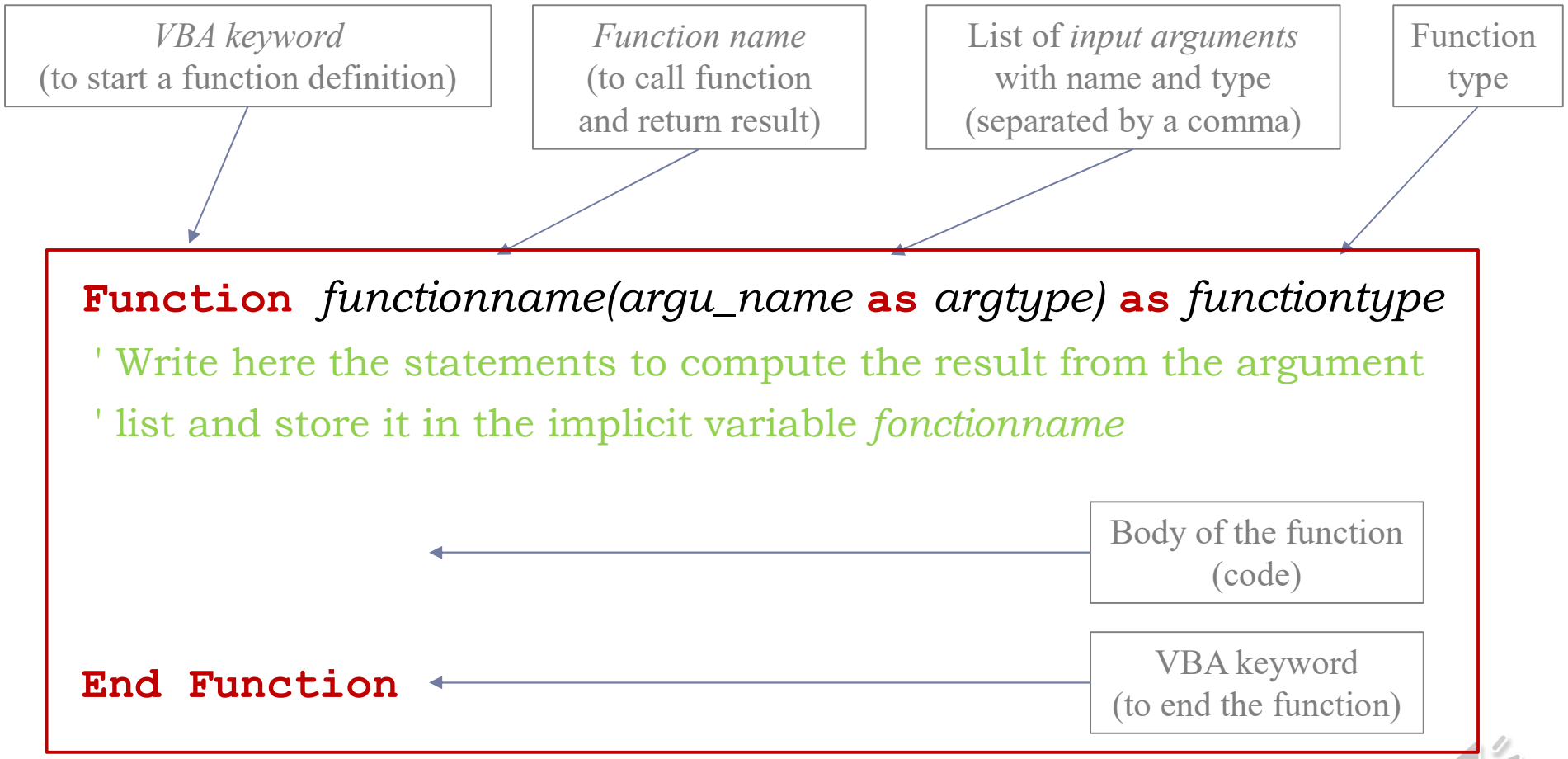
Examples:

```
Function SurfCarre(L As Double) As Double
  SurfCarre = L * L
End Function


Function SurfRect(H As Double, W As Double) As Double
  SurfRect = H * W
End Function


Function SurfCercle (Rayon As Double) As Double
 Const Pi = 3.14159265358979
 SurfCercle = Pi * Rayon * Rayon
End Function
```

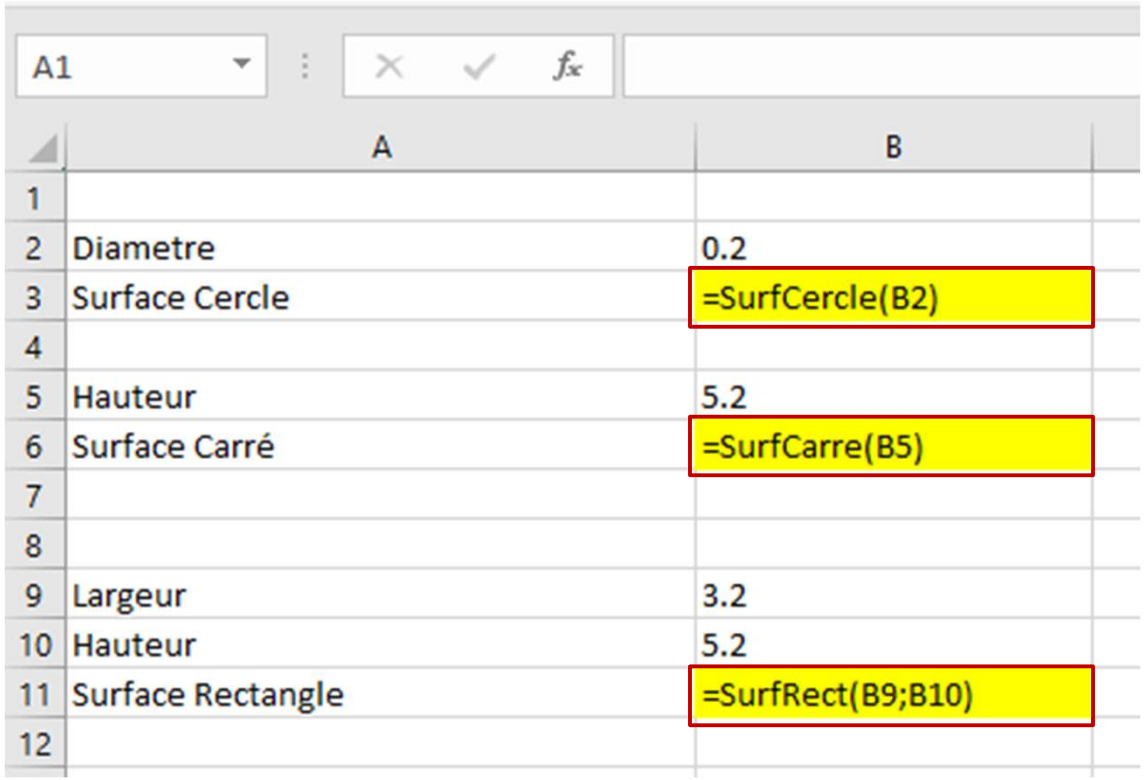
« Local » variable

Functions

Syntax:



Calling your function from Excel



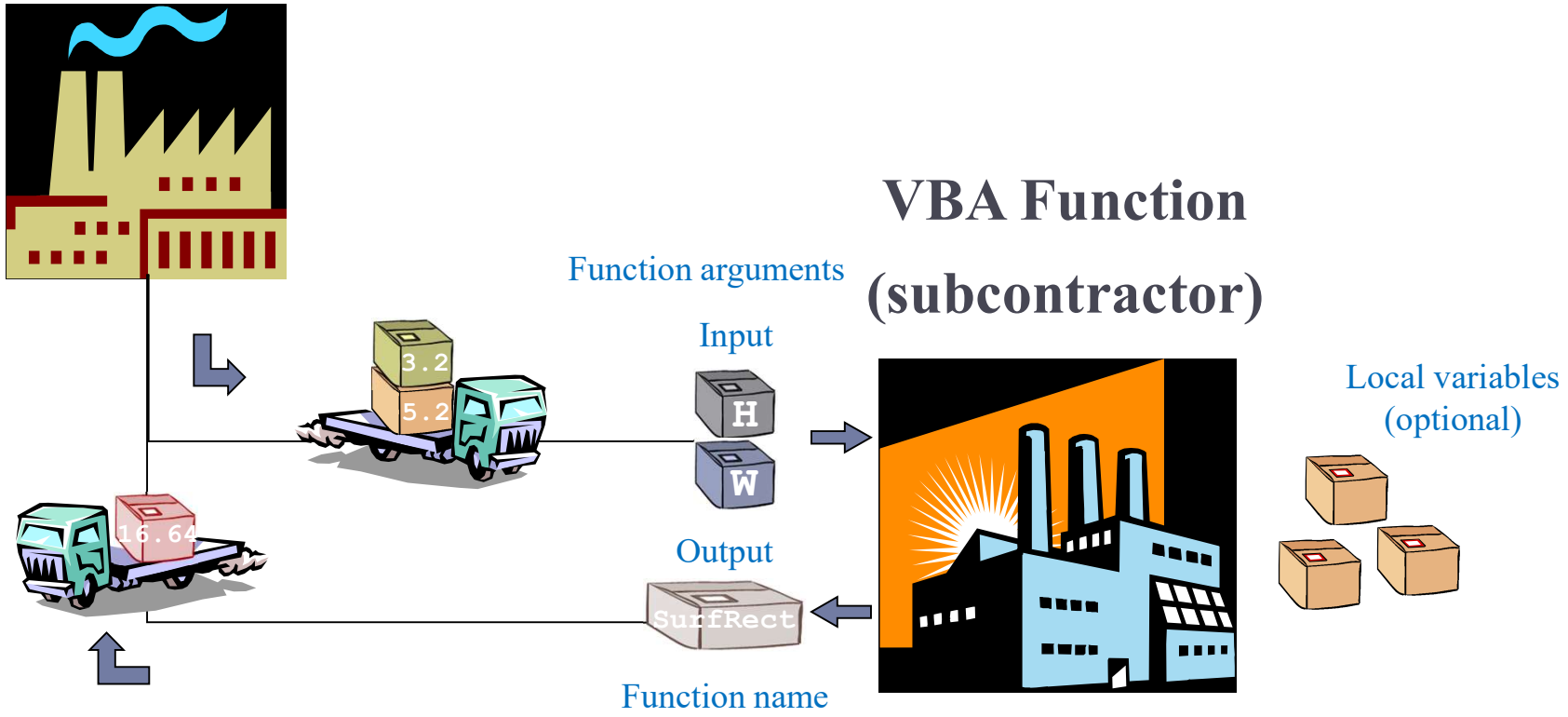
	A	B
1		
2	Diametre	0.2
3	Surface Cercle	=SurfCercle(B2)
4		
5	Hauteur	5.2
6	Surface Carré	=SurfCarre(B5)
7		
8		
9	Largeur	3.2
10	Hauteur	5.2
11	Surface Rectangle	=SurfRect(B9;B10)
12		

Note that the argument separator in Excel is a semi-colon ";" (while it is a colon "," in VBA)

Variable transfer with function

9	Largeur	3.2
10	Hauteur	5.2
11	Surface Rectangle	=SurfRect(B9;B10)

Excel (factory)




```
Function SurfRect(H As Double, W As Double) As Double
  SurfRect = H * W
End Function
```

Creating Procedures in VBA

Two types of procedures

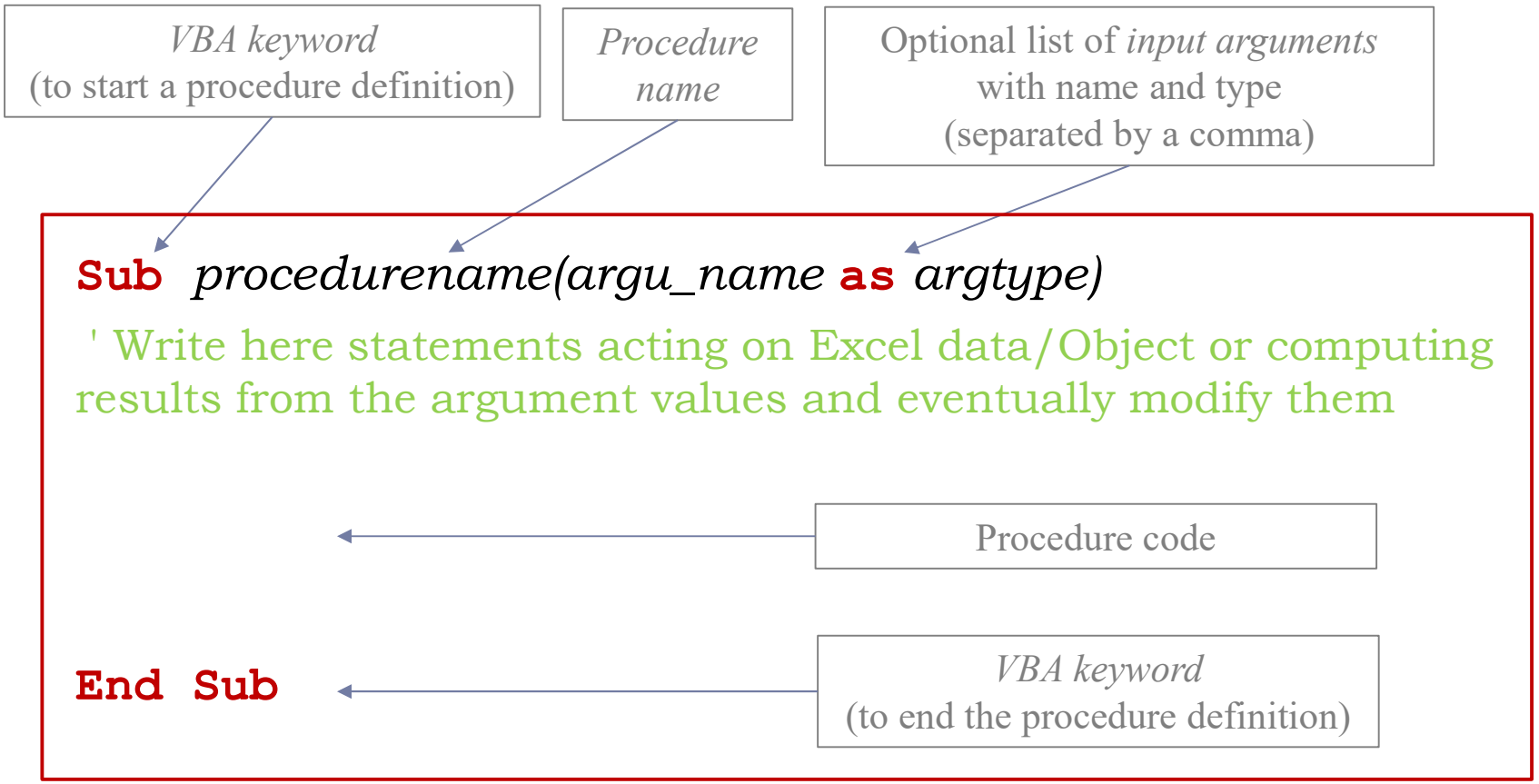
Like *functions*, *procedures* allow in automating operations.

It differs from *functions* because :

- It does not necessarily required arguments
- It can return more than one computed result
- It can handle any type/size of arguments (including matrix)
- It can modify the values of its arguments
- You call a procedure with a **`Call`** instruction

A **macro** is a special procedure which does not have any argument. It will act autonomously on Excel and Windows data. In Excel, you can call it from the *Tools/Macro* menu.

Procedure syntax

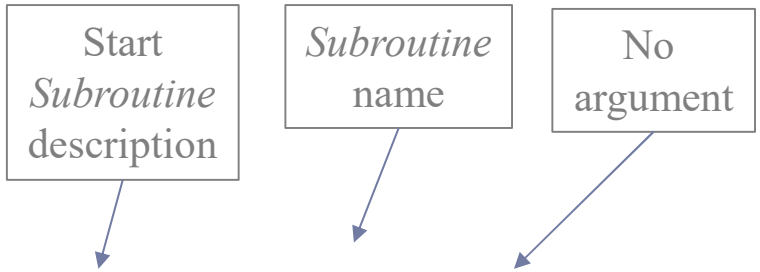


Call Syntax:

```
Call procedurename(argu_values)
```

Subroutine example

The following subroutine ask for a diameter value and display the corresponding area.



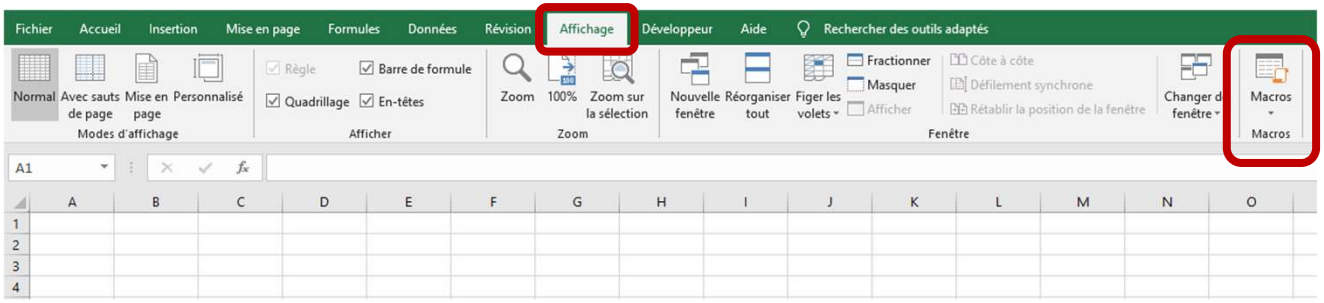

```
Sub CircleArea()
  Dim D as Double
  ' User value is be stored in D variable
  D = InputBox("Diametre ?", "Saisie",2.0)
  MsgBox(" Circle Area : " & string(3.14/4*D^2))
End Sub
```

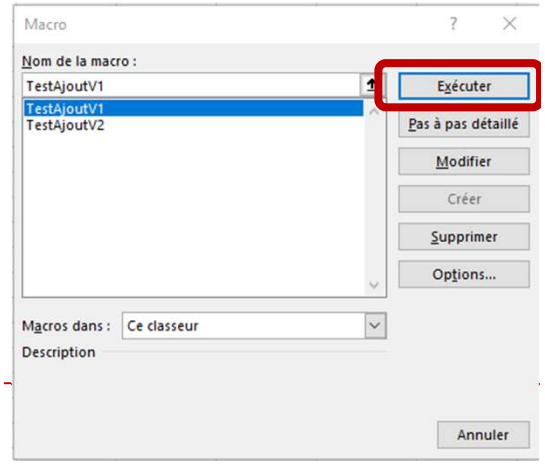
Running your macro

- From VBA code
  - `Call TestAjoutV1()`

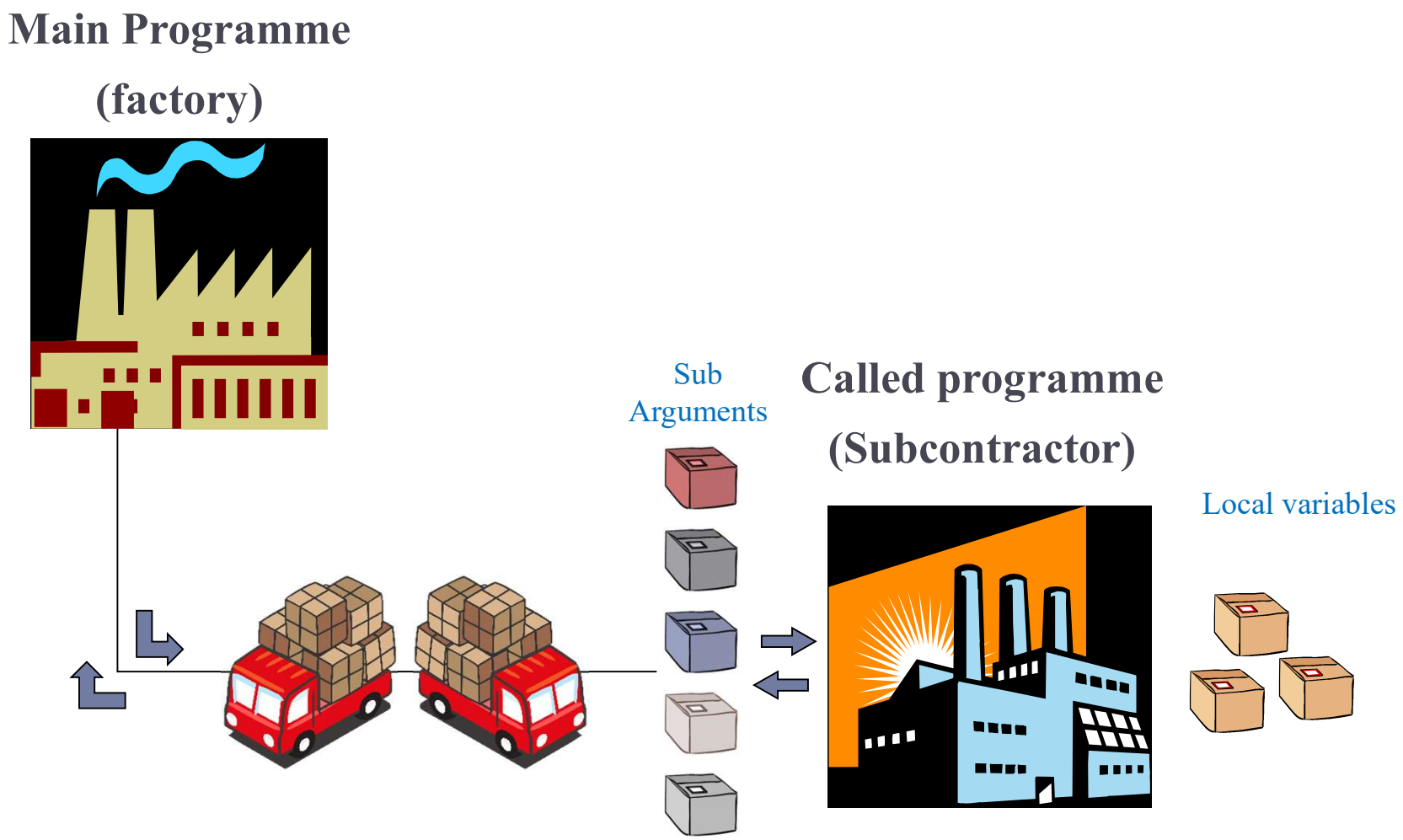
- From VBA Editor
  - Run the current Sub (depend on cursor position)

- From Excel
  - Use the Macro Menu





Variables transfer in VBA



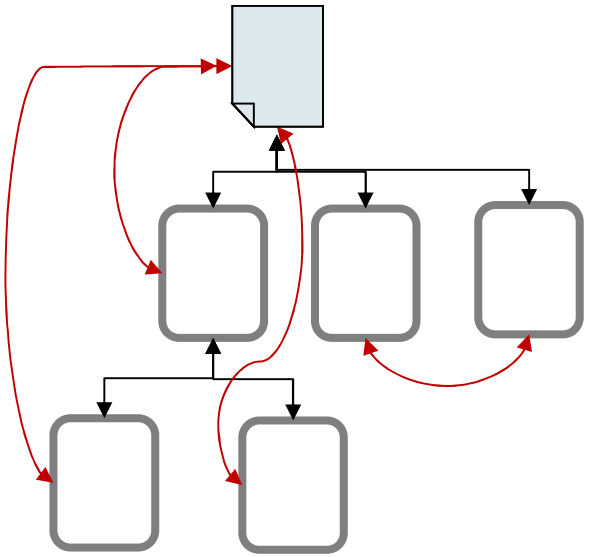
No distinction between Input and Output arguments

Visibility under VBA

- Function/Procedure visibility
- Variable visibility

Visibility

- A VBA programme is usually made of several **procedures** (Sub) and **functions**. VBA has to "*find*" them when they are called.



- Managing **variables**: some variables in procedure may be required for computation in other procedures, while other variables are used only for a single computation.

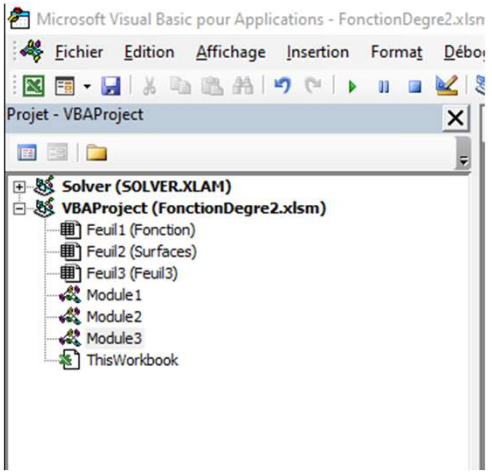
Is it essential to understand how information is transferred from one part of the programme to another.

It often explains a lot of errors of beginners.

VISIBILITY OF FUNCTIONS/SUBS

Visibility of functions and procedures

Write your functions/Sub in Modules



Use **`Private`** and **`Public`** to control the visibility.

`Private`:

it indicates that the procedure is **accessible only to other procedures in the module** where it is declared.

e.g. : `Private Function SurfCarre(L As Double) As Double`

`Public` (default):

It indicates that the procedure is accessible **to all other procedures in all modules**.

e.g. : `Public Sub TestAjout()`

Visibility of functions/subroutines

Module A

```
Public Sub Sa1()
…
End Sub


Public Sub Sa2()
…
End Sub


Private Function Fa1(x as double)
…
End Sub
```

Module B

```
Private Sub Sb1()
…
End Sub


Private Sub Sb2()
…
End Sub


Private Function Fb1(y as double)
…
End Sub
```

Fa1 can only be used in **Module A only**

Sa1 and Sa2 can be used in **Modules A and B**

Sb1, Sb2, Fb1 can only be used in **Module B only**

(Sb1, Sb2, Fb1 can NOT be used in Module A)

VISIBILITY OF VARIABLES

Variable visibility

Local Variables

In **functions** and **procedures**, all variable declared inside the function are local variables. They can only be used inside the function.

You even can declare a function or procedure as **static** to keep values of local variable in memory between two calls.

Information transfer between functions and Sub

In **functions**, Arguments are received during the function call and the name of the function is used to return the result to the calling programme.

In **procedures**, arguments are received **and** returned during the sub call.

Global variables/constants

- **Global variables/constants**
  - They are used to share information between programme parts.
  - Principe : declare them in the **declaration section**
  - Syntaxes:

```
Dim      …   As   …      (shared in the same module)
Public   …   As   …      (shared in all the project)
```

Global variable name

Global variable type

Global variables/constants

In the following example, Pi a is **global** constant which can be used in any part of your programme:

```
' Declaring Pi as global variable
Public Const Pi = 3.14159265358979

Function SurfaceCercle (Rayon As Double) As Double
  SurfaceCercle = Pi * Rayon^2            ' Use Pi anwhere
End Function

Function VolumeSphere(Rayon As Double) As Double
  VolumeSphere = 4/3*Pi * Rayon^3         ' Use Pi anwhere
End Function
```

Visibility of variables

Module A

```
Dim AA as double
Public BB as double


Function Fa1(a as double) as double
…
End Function


Sub Sa1()
…
End Sub


Sub Ra2(b as double)
…
End Sub
```

Module B

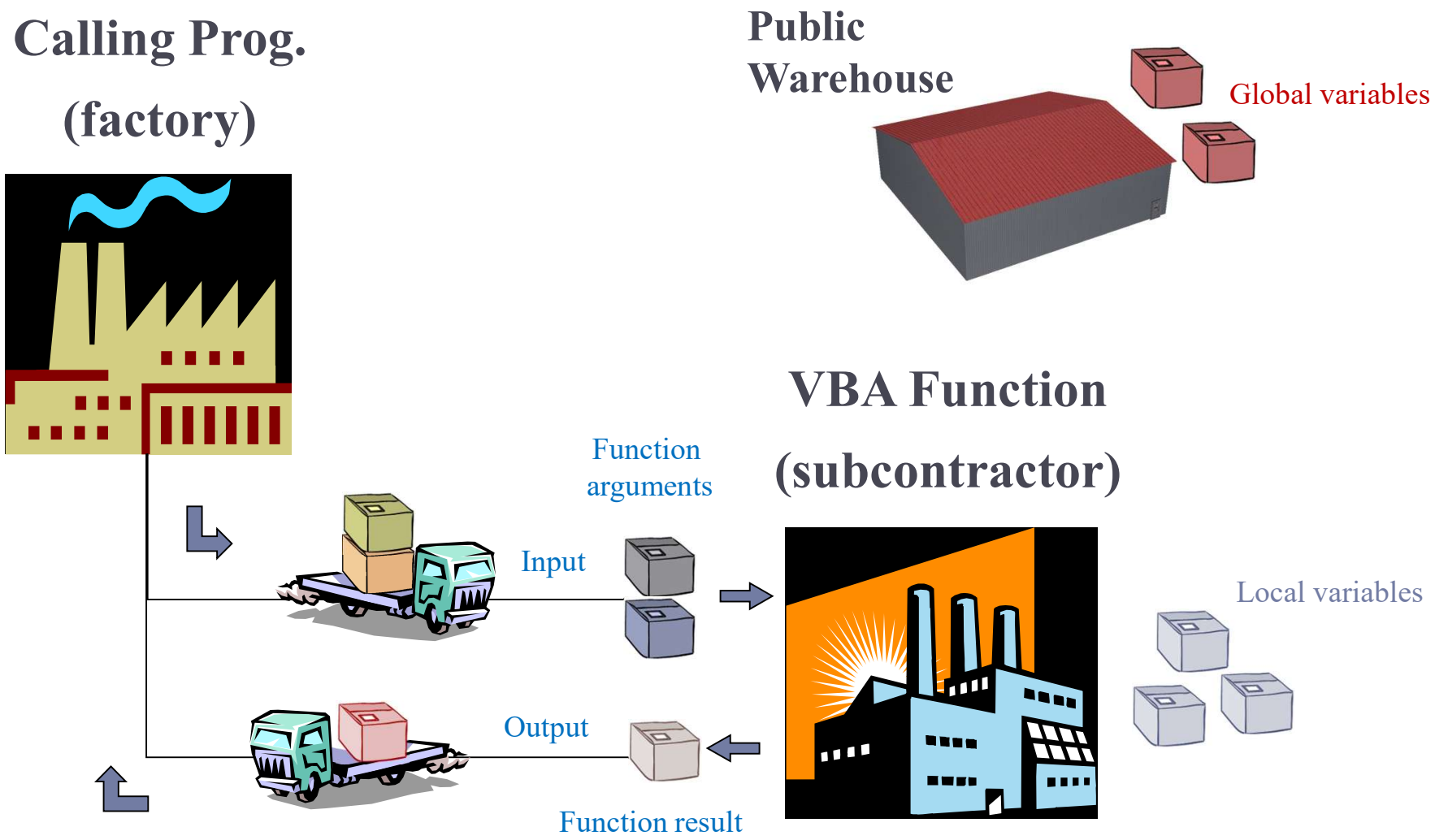
```
Dim XX as double
Public YY as double


Sub Sb1(a as double, b as double)
…
End Sub


Function Fb1(c as double) as double
…
End Function


Function Fb2(d as double) as double
…
End Function
```

AA can be used in Fa1, Sa1 and Ra2 but not in Sb1, Fb1, Fb2

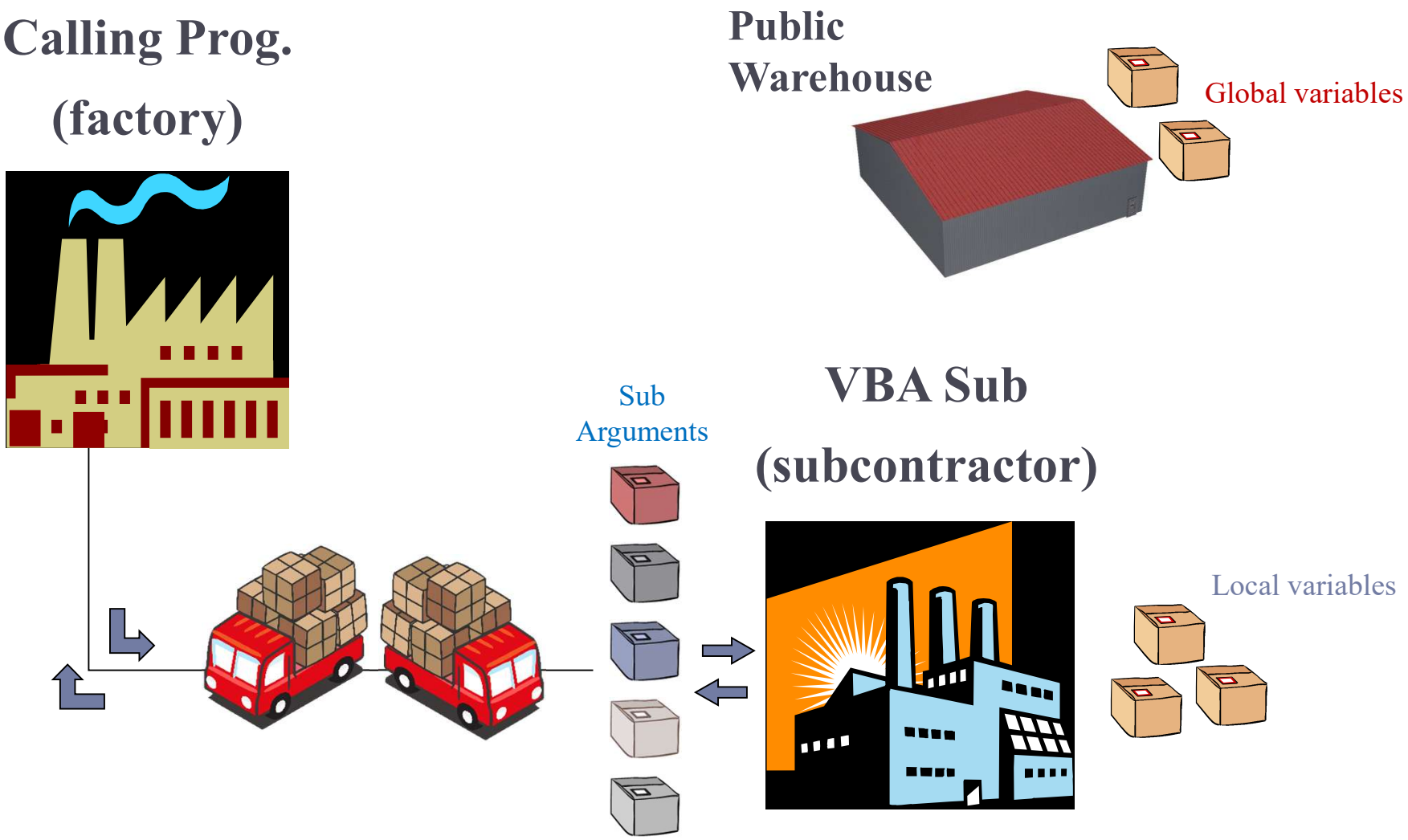
XX can be used in Sb1, Fb1, Fb2 but not in Fa1, Sa1, Ra2

BB and YY can be used **in all functions and subs**

Variable transfer with function



Variables transfer with Sub



No distinction between Input and Output arguments

Flow controls

Controlling your programme

A VBA programme consists in a list of statements which are sequentially read from top to bottom where each line is interpreted from right to left.

VBA proposes some additional *keywords (called Controls)* to allow:

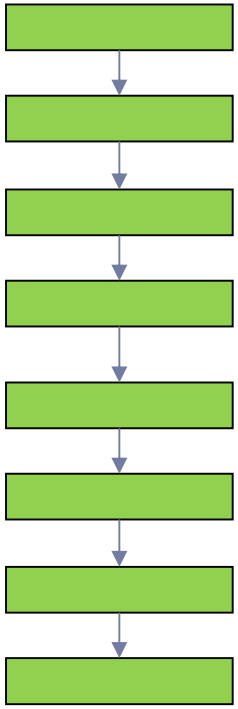
- repeating some group of the statements (LOOP)

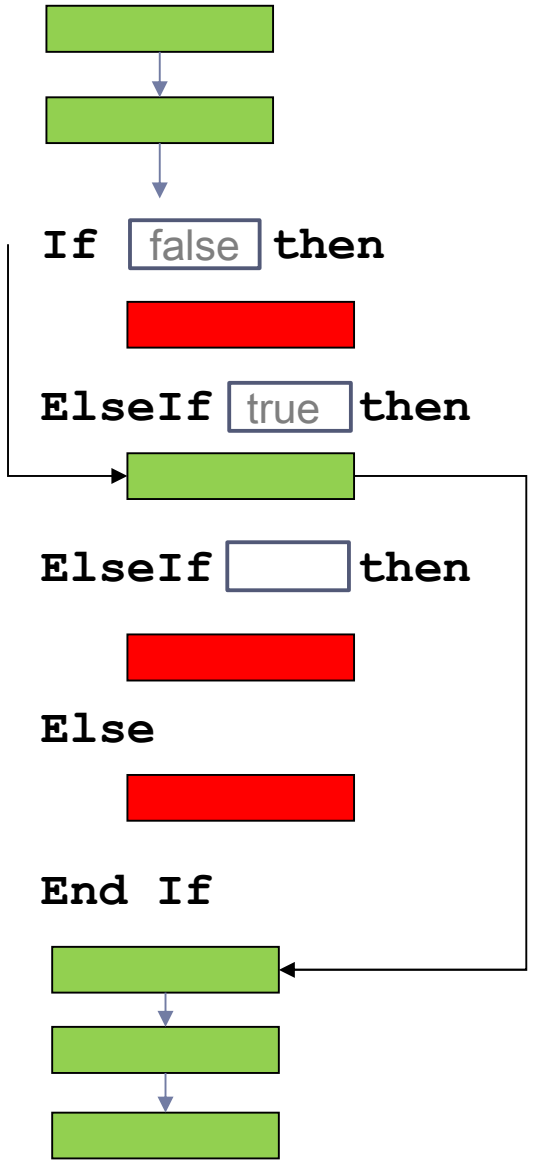
or

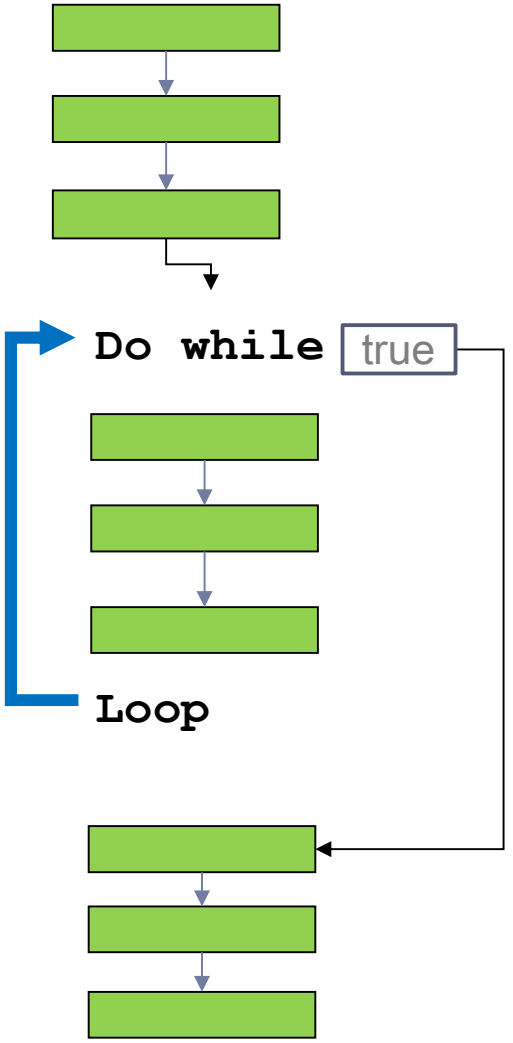
- to conditionally run some part of the programme (TEST).

Controlling your programme

Simple programme







Flow controls

In VBA, there are 4 elementary flow control statements

- **TEST** to executes a group of statements based on some conditions.
  - **if** /**end If** based on logical conditions.
  - **Select case** / **End Select** based on some value conditions

- **LOOP** to repeat a group of statements
  - **Do while** / **Loop** based on some logical conditions.
  - **Do** / **until** based on some logical conditions.
  - **For** / **Next** for a fixed number of times.

Many structures can be left prematurely with the **Exit** key word The

- **Exit For** terminates execution of a for loop.
- **Exit Do** terminates execution of a while/until loop.
- **Exit Function** terminates execution of a function.
- **Exit Sub** terminates execution of a Sub.

Flow controls

Each keyword is associated to a list of statements which MUST BE delimited by the appropriate keyword

TEST

```
If xxx
   statement
End If
```

```
Select Case xxx
   statements
End Select
```

REPETITION

```
Do while xxx
   statements
Loop
```

```
Do
   statements
Loop while xx
```

```
Do Until xx
   statements
Loop
```

```
Do
   statements
Loop Until xx
```

```
for i = xxx
   statements
Next i
```

IF

Choice Structure : "If"

Full syntax :

If **logical_expression** ***Then***

statements 1

Else If **logical_expression**

statements 2

Else If **logical_expression**

statements 3

Else

default statements

End If

a test yielding a Boolean result *True* (1) or *False* (0).

Examples:

```
x < 2
0<=z AND z<=10
A = 0
```

Shorter syntax :

If-End If

If-Else-End If

SELECT

Choice structure : “Select"

Select Case ***variablename*** → Variable with finite value

Case **value1** → List 1 of possible value(s)

statement list 1

Case **value2** → List 2 of possible value(s)

statement list 2

Case **value 3** → List 3 of possible value(s)

statement list 3

Case Else → when all previous cases failed

default **statements**

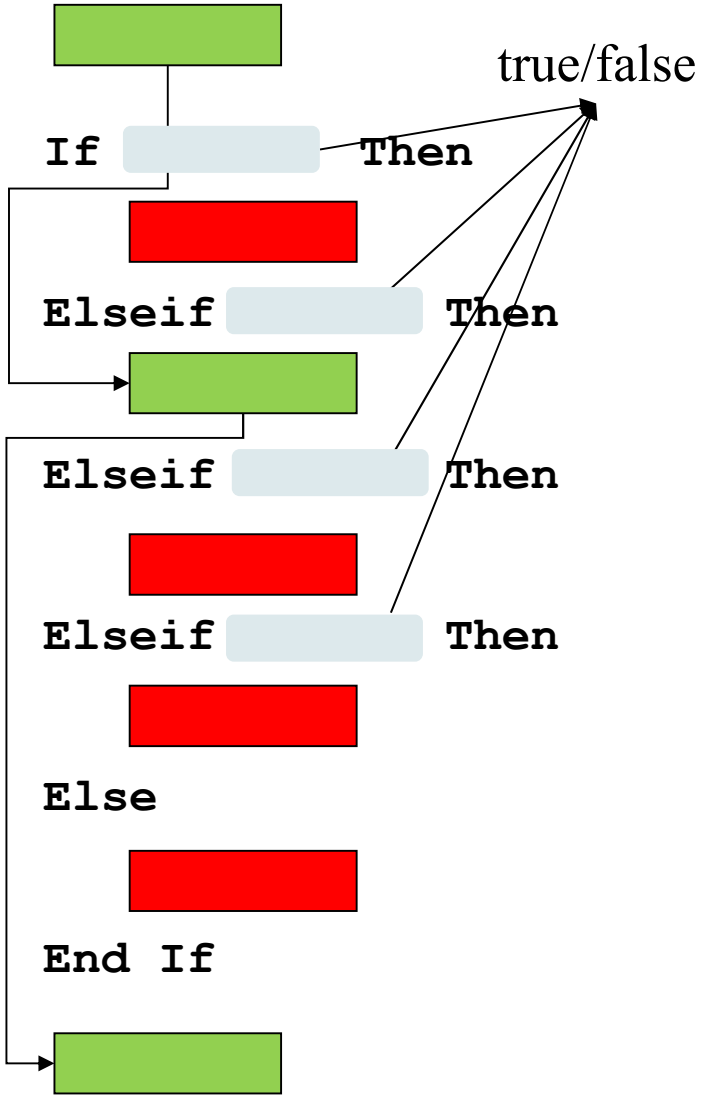
End Select

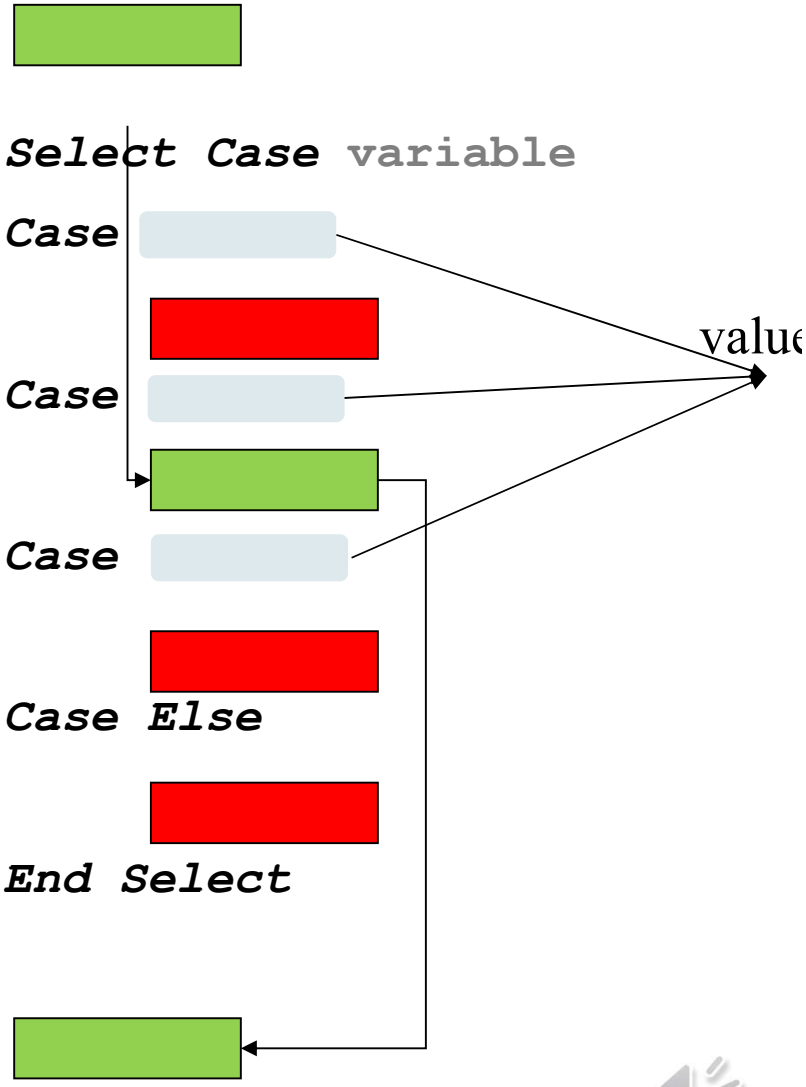
Use: `Case 1 To 4` **to assess:** ***variablename =1, 2, 3 or 4***

`Case 5,8,9` **to assess:** ***variablename =5, 8 or 9***

`Case Is <=10` **to assess:** ***variablename ≤10***

Debbuging a choice structure





DO WHILE/DO UNTIL

Repetition Structure : “Do While/ Do Until"

Example : ask for a value to the user
until it satisfies a given condition

```
n = InputBox "Donnez un nombre entre 1 et 10 : "
Do while n<1 OR n>10
    MsgBox("Réponse incorrecte")
    n = InputBox(" Donnez un autre nombre : ")
Loop
```

```
Do
   n = InputBox("Donnez un nombre entre 1 et 10 : ")
Loop While n<1 OR n>10
```

```
Do
   n = InputBox ("Donnez un nombre entre 1 et 10 : ")
Loop Until n>=1 AND n<=10
```

Structure “Do while + Exit Do"

Exiting from a loop

Syntax :

```
Do while test1
    statements
    if  Test2
        statements
        Exit Do
    end
    statements
Loop
```

This block allows for leaving the loop according to a second condition (Test2)

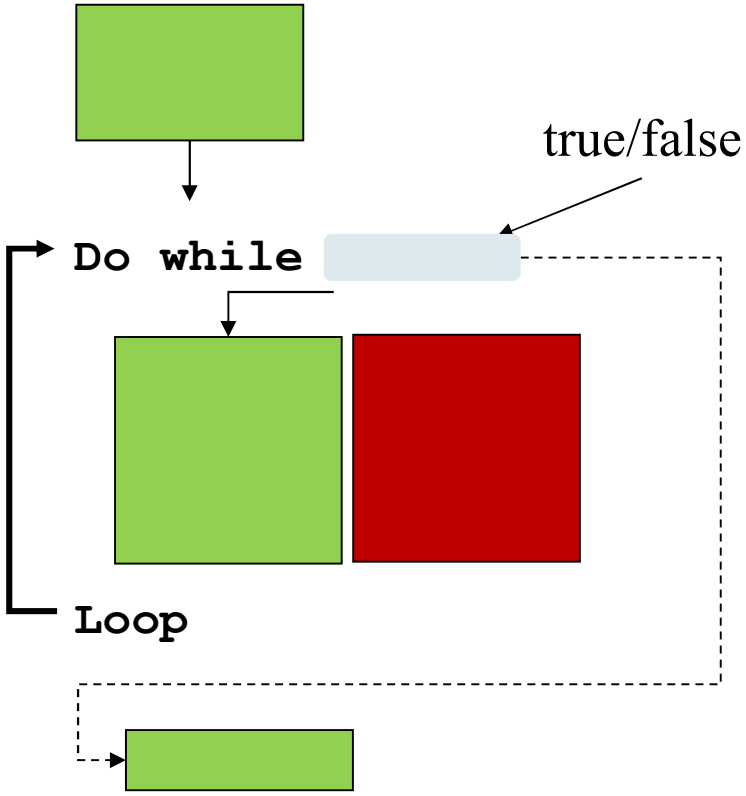
Repetition structure : "for"

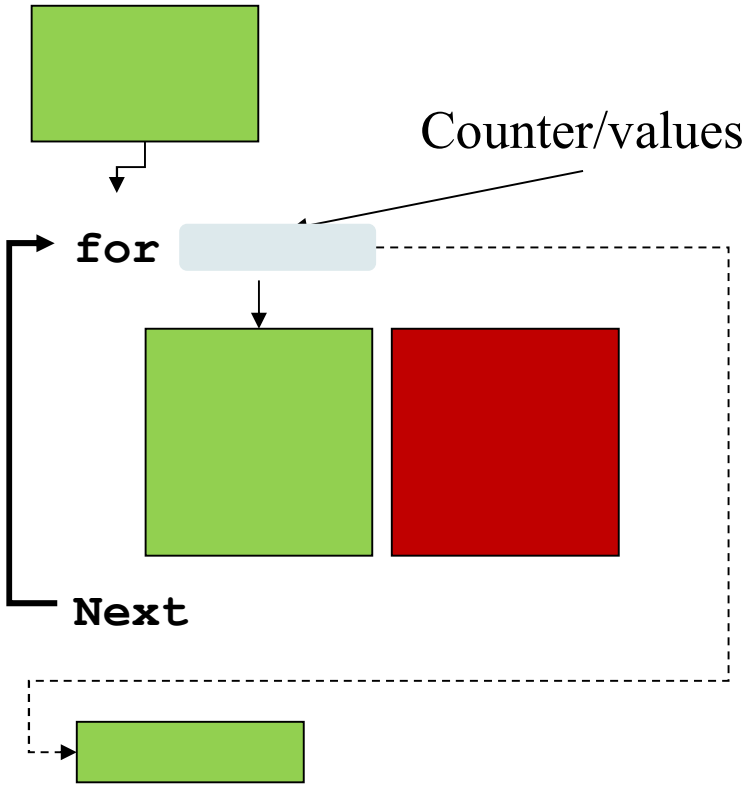
Syntax :

For *counter* = *startvalue* **To** *endvalue* [**Step** *increment*]
statements
Next *counter*

Step *increment* is optional

Debbuging a repetition structure



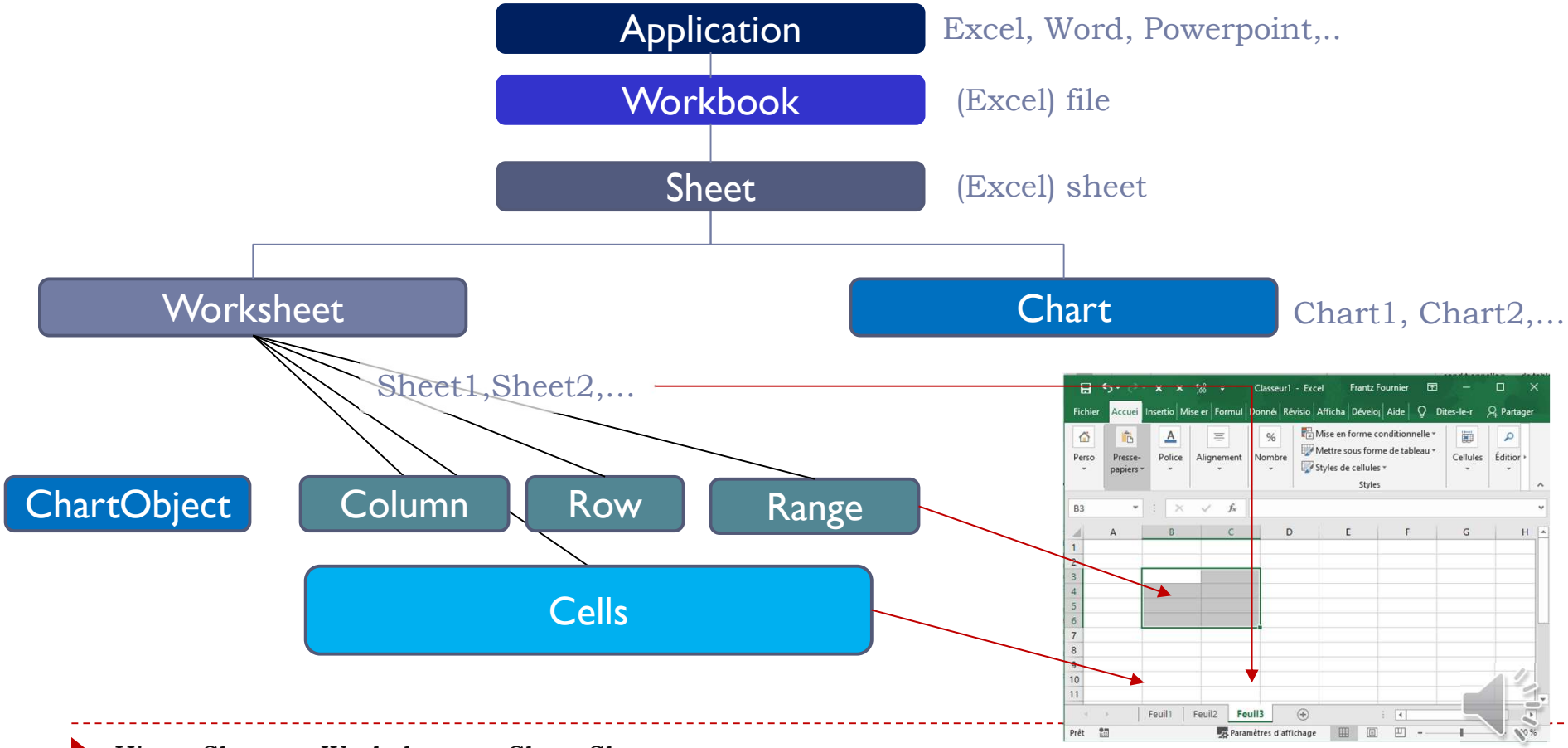


Handling Excel cells with VBA..

Excel **Objects**

VBA can be used to handle Microsoft "objects". Objects depends on the **Application** used (Word, Excel, Powerpoint,…).

- In **Word**: objects are Documents, Chapter, Words,…
- In **Excel**: objects are Files (**`Workbooks`**), Sheets (**`WorkSheets`**), Cells (**`Cells`**), Cell ranges (**`Range`**) and Charts (**`Charts`**)



Hints: Sheets = Worksheets + Chart Sheets

Referring to Excel Objects

- By **Index** in a collection

`Workbooks(1)`	1st open Excel file
`Worksheets(3)`	3rd Excel Sheet
`Columns(1)`	1st col. or `Columns("A")`
`Rows(3)`	3rd rows
`Cells(2,4)`	Cell at line 2 and column 4

- Using a **Range** of cells

`Range("A1","B2")`	Range of 4 cells (from cell A1 to cell B2)
`Range("A1:B2")`	The same range
`Range(Cells(1,1),Cells(2,2))`	The same range
`Range("A:A")`	Whole A column
`Range("A:C")`	Columns A to C
`Range("1:3")`	Lines 1 to 3
`Range("A:B,D:D")`	Columns A, B, D (excluding C)

- By **name** (for files and sheets)

```
Workbooks("Test.xlsm")
Worksheets("Mesures")
```

Example of complete reference to a cell:

```
Workbooks("Test.xlsm").Worksheets(2). _
   Cells(2,3)

Workbooks(3).Worksheets("Data"). _
  Cells(2,3)

  Workbooks("Test.xlsm"). _
     Worksheets("Data").Cells(2,3)
```

Referring to Excel Objects: Example

```
Sub SDegre2_from_Excel_Cells()
Dim a As Double, b As Double, c As Double
Dim  x1 As Double, x2 As Double
Dim delta As Double
Dim Erreur As Boolean

Erreur = False

' Retrieve a, b, c values from Excel
cells
a = Worksheets("Coeff").Cells(4, 2)
b = Worksheets("Coeff").Cells(5, 2)
c = Worksheets("Coeff").Cells(6, 2)

delta = b ^ 2 - 4 * a * c

' Test discriminant and compute the 2
real solutions
If delta > 0 Then
  x1 = (-b - Sqr(delta)) / (2 * a)
  x2 = (-b + Sqr(delta)) / (2 * a)
ElseIf delta = 0 Then
  x1 = (-b) / (2 * a)
  x2 = x1
Else
  MsgBox ("No solution in Re")
  Erreur = True
End If

' Write results to Excel cells
If Not Erreur Then
   Worksheets("Coeff").Cells(4, 5) = x1
   Worksheets("Coeff").Cells(5, 5) = x2
Else
   Worksheets("Coeff").Cells(4, 5) = ""
   Worksheets("Coeff").Cells(5, 5) = ""
End If

End Sub
```

Declaring Excel Objects in VBA

Same syntax for **declaration**: **Dim** … **As** …

Examples:

```
Dim   Fich   As Workbook
Dim   Feuil  As Worksheet
Dim   Data   As Range
Dim   Fig    As Chart
Dim   Obj    As ChartObject
```

But **initialisation** with dedicated syntax :

Set *variablename* **=** *Objectname*

Examples:

```
Set Fich  = Workbooks("Essai.xlsm")
Set Feuil = ActiveWorkbook.Sheets(1)
Set Data  = Range("A1:D5")
Set Data  = Range("A:A")          ' All the A column
Set Data  = Selection             ' Current selection
```

Object properties

Each object has properties to describe them as far as **contain** (*value*) and **form** (*foreground and background colors, font size and type,…*) are concerned.

Some examples:

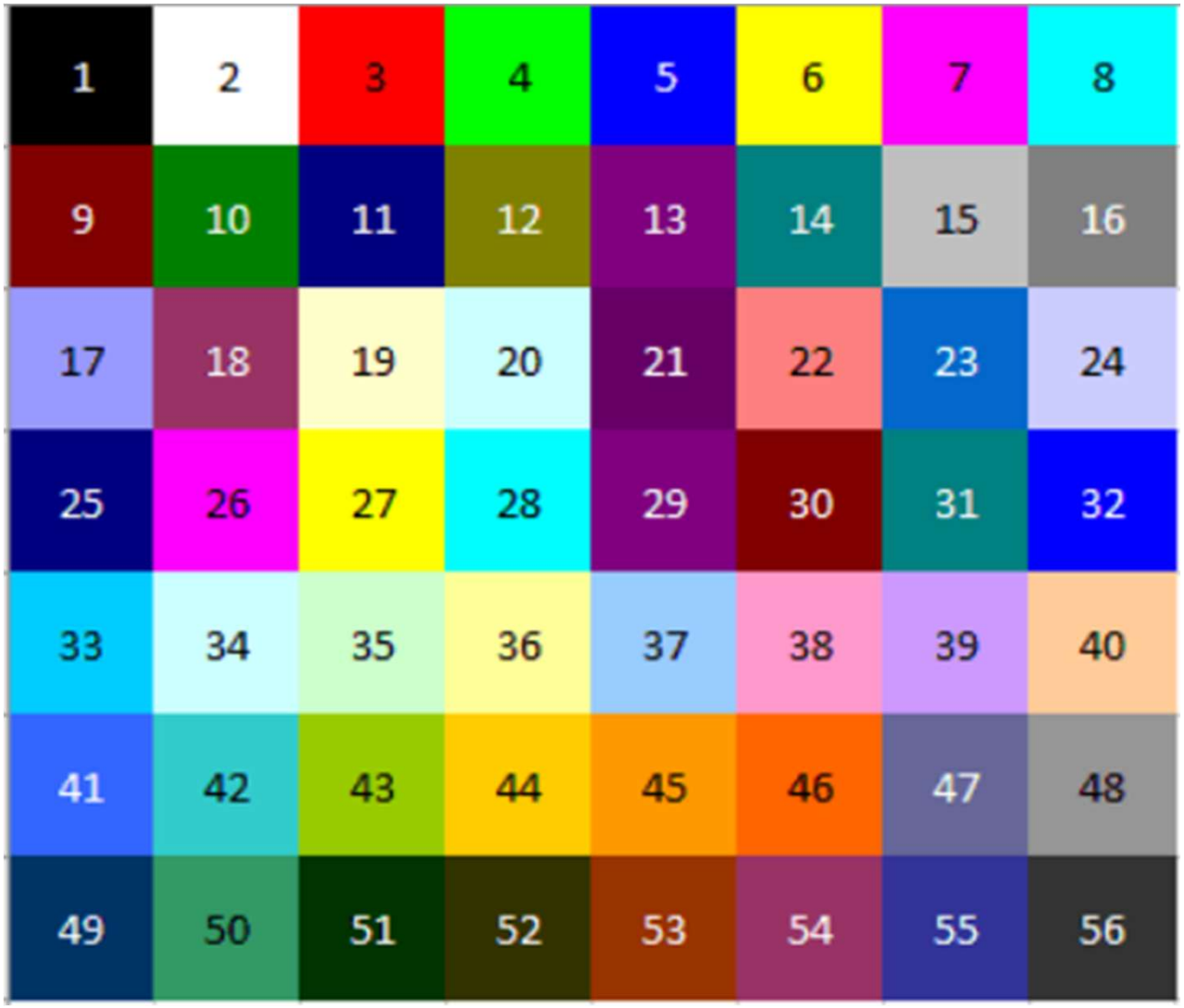
```
Cells(2,3).value                ‘ Value in the cell at row 2 and column 3
Cells(2,3).font.bold            ‘ Boldtype of the font in the cell
                                ‘ (its value is either False or True)
Cells(2,3).font.colorindex      ' color number of the cell
Cells(Ligne,Col).Comment.Text   'Text of the cell comment
```

Applications:

```
Cells(2,3).value = 2
N = Cells(1,5).value
For I = 1 To 10
  Cells(I,3).value = Cells(I,2).value +10
Next I
```

```
dernièreLigne = Cells(Rows.Count, 1).End(xlUp).Row
```

Codification des couleurs



Object methods

Each object is also associated with **method** for some predefined actions.

Some examples:

```
Workbooks"Classeur1.xlsm").Activate
```
activate the Excel file *Classeur1.xlsm*

```
Workbooks("Classeur1.xlsm").Open
```
open an existing file

```
Workbooks("Classeur1.xlsm").Close
```
close the file

```
Worksheets(3).activate
```
activate 3rd sheetin current Excel file

```
Worksheets.add
```
add a new sheet

```
Worksheets("Feuil3").delete
```
delete a sheet named "*Feuil3*" (irreversible action !!)

Calling Excel function from VBA module

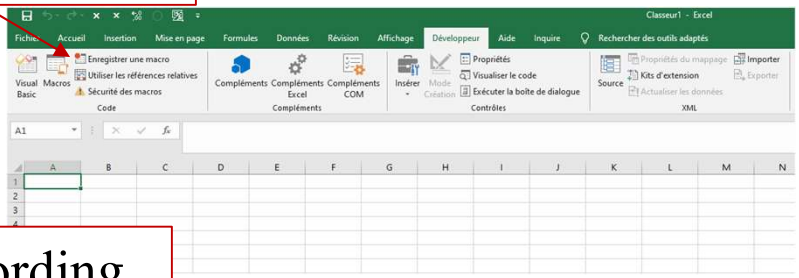
```
Dim Mesures as Range
Dim NbLig As Long, NbBlanc As Long, NbVal As Long
Dim Pi as Double
Dim Som As Double, Moy As Double, Ecart As Double

Worksheets(3).activate ' activate 3rd sheet in the Excel
file
Set Mesures = Range("A:A") ' define a range as the 1st
column

' Retrieving the value of Pi
Pi = Application.WorksheetFunction.Pi()

' Computing Sum, Mean or Standart Deviation
Som = Application.WorksheetFunction.Sum(Mesures)
Moy = Application.WorksheetFunction.Average(Mesures)
NbVal = Application.WorksheetFunction.CountA(Mesures)
```

MACRO RECORDING

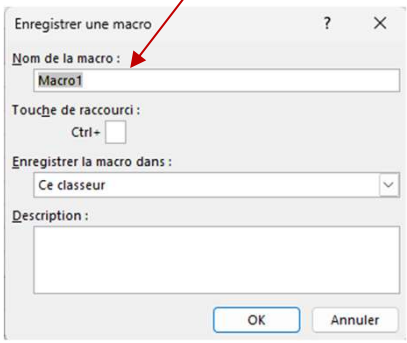
Macro recording

Excel-VBA can record all your actions in Excel to convert them into a series of statement in a VBA macro.

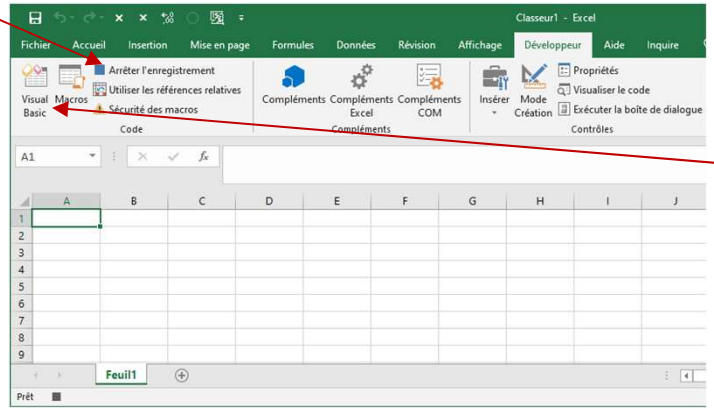
1-Start recording



2- Give a name to your macro



3-Stop recording



4-Look at recorded VBA code

It is a simple way to learn how to handle Excel objects with VBA

AND MANY OTHERS ACTIONS…