Excel
VBA programming
Visual Basic for Applications

Herve Hocquard

http://www.labri.fr/perso/hocquard

Writing and using custom functions in Excel

CUSTOM FUNCTIONS

A custom function is a VBA function that can be called in an Excel workbook. It takes information from the workbook sheets as input (mostly) and returns a value inserted into a cell (most often as well).

```
Formalism Function FunctionName(settings) As type of data
```

Is an identifier which must respect the VBA syntax

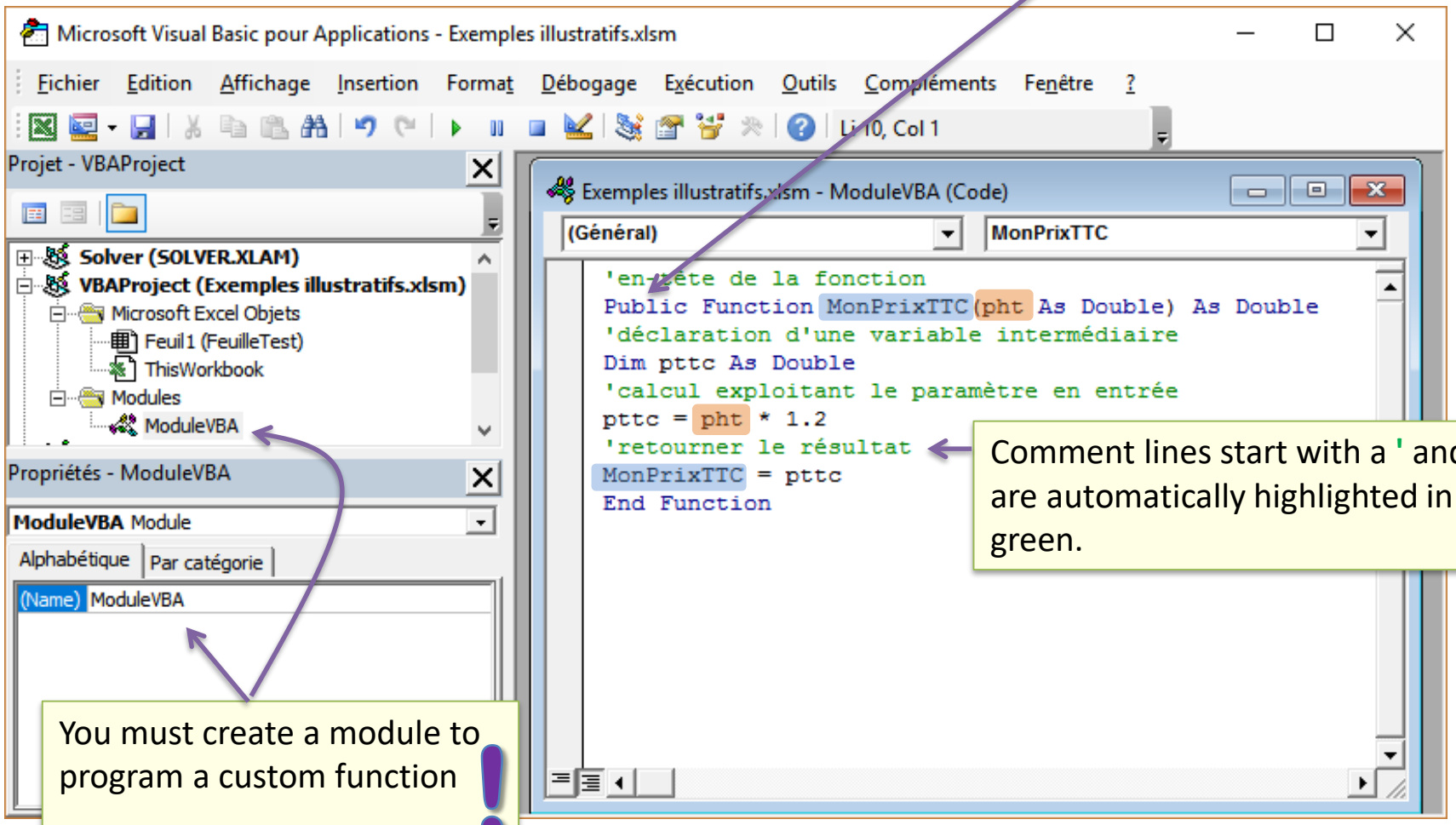
Type of the value returned by the function.

The information that the function takes as input takes the form *parameter_name As parameter type*. There may be several, they are separated by" , " in that case.

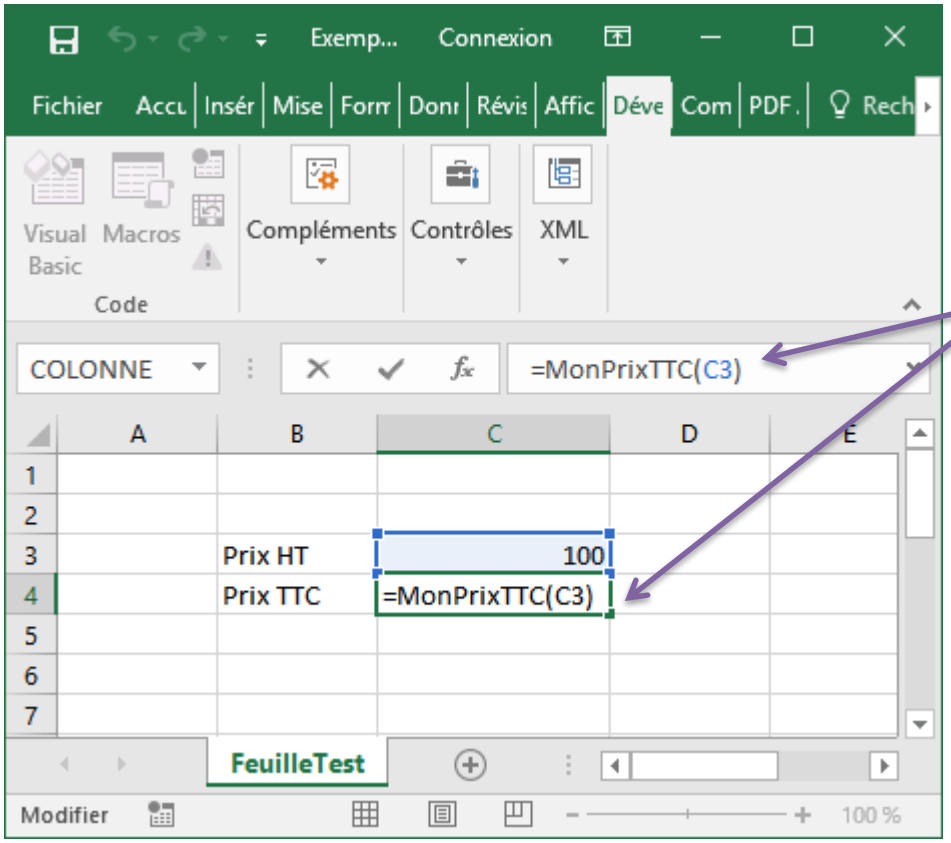
An Excel workbook containing VBA code should be saved in XLSM format, supporting macros. Otherwise you lose your code.

Entry: price excluding VAT (real number)
Output: price including VAT (real number)

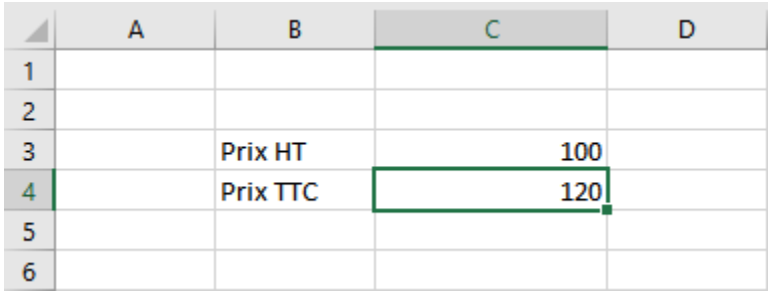
Public so that the function is visible outside the module, especially in the spreadsheet



Using the function in an Excel sheet



The function can be inserted into the worksheet like any other Excel function. It is accessible in the category "Custom functions ".

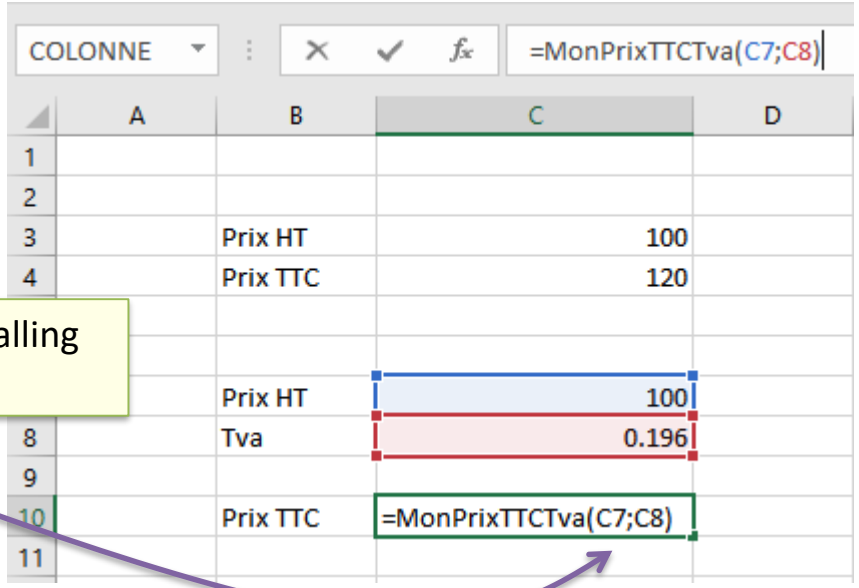


The result is displayed once the function has been inserted and validated. The function is automatically called each time the sheet needs to be recalculated (like other standard Excel functions).

Entries: price excluding VAT (real n.), VAT (real n.)
Output: price including VAT (real n.)

(1) The parameter separator is the "," when defining the function.

```
'function with 2 parameters
Public Function MonPrixTTCTva(pht As Double,VAT As Double) As Double
'declaration of an intermediate variable
Dim pttc As Double
'calculation using the input parameters
pttc = pht * (1 + VAT)
'return the result
MonPrixTTCTva = pttc
End Function
```



(2) Very strangely, it becomes ";" when calling the function in the spreadsheet.

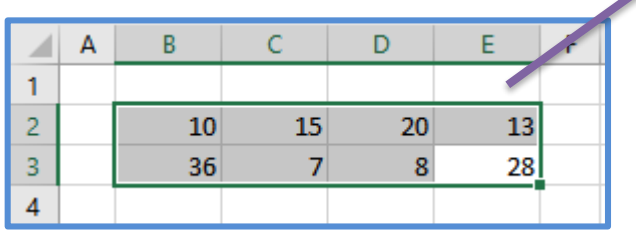
USE NATIVE EXCEL FUNCTIONS

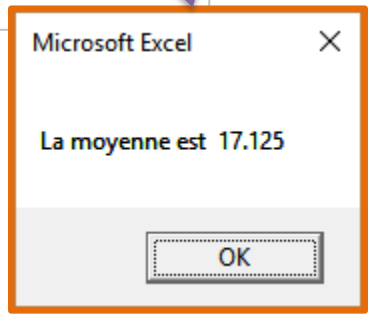
Excel has powerful native functions. We can access it in our VBA programs.

Example: Check that a selection is simple (only one area), then calculate and display the average of the values in a dialog box.

```
Sub MyMeanSelection()
'var. intermediate
Dim Average As Double
'check selection
If (Selection.Areas.Count > 1) Then
 MsgBox ("Be careful, this is not a simple selection")
Else
 '' have Excel calculate the average of the selection
 average = Application.WorksheetFunction.Average(Selection)
 MsgBox ("The mean is" & Str(average))
End If
End Sub
```

Note the syntax. !





Further with programming ...

ALGORITHMIC STRUCTURES

Allows you to activate part of the code depending on whether or not a condition is met.

Syntax

```
If condition Then
 instruction block
 if the condition is true
Else
 instruction block
 if the condition is false
End If
```

(1) Condition is often a comparison operation

(2) The return value of Condition is of type boolean (True or False)

(3) Then must be on the same line as If

(4) The part Else is optional (do nothing if the condition is false)

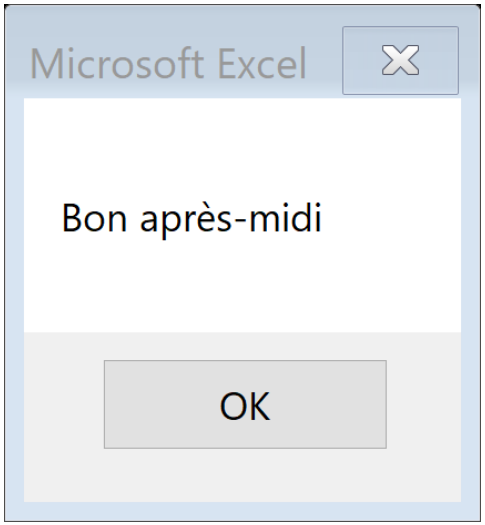
(5) It is possible to nest another conditional structure If in the instruction blocks

Entries: price excluding VAT (real n.), product category (string)
Output: price including VAT (real n.)

```
Public Function MyTTCBis(pht As Double, cat As String) As Double
'declare the calculation variable
Dim pttc As Double
'depending on the product category
If (cat = "luxury") Then
 pttc = pht * 1.33
Else
 'the value of cat is different from' luxury '
 pttc = pht * 1.2
End If
'return the result
MyTTCBis = pttc
End Function
```

Conditional branching IF /ElseIf - An example

```
Sub bonjour()

Dim msg As String

If Time < 0.5 Then

    msg = "jour"

ElseIf Time < 0.75 Then

    msg = "après-midi"

Else

    msg = "soir"

End If

MsgBox "Bon" & msg


End Sub
```



Used to activate a part of the code according to the values taken by a control variable. Can substitute for IF, but not always, it all depends on the form of the condition (*compound condition, we must go through an IF*).

Syntax

```
Select Case variable
 Case value 1
 instruction block
 Case value 2
 instruction block
 ...
 Case Else
 instruction block
End Select
```

(1) Variable is the control variable, it can be of any type in VBA, including a real or a string

(2) Value must be of a type compatible with variable

(3) The part Case Else is optional

(4) Nesting with another IF or another Select Case (another control variable) is possible.

Multiple branch SELECT CASE - An example

Entries: price excluding VAT (real n.), product category (string)
Output: price including VAT (real n.)

```
'select case function
Public Function MyTTCAccording(pht As Double, cat As String) As Double
'declare the calculation variable
Dim pttc As Double
'depending on the product category
Select Case cat
 Case "luxury"
 pttc = pht * 1.33
 Case Else
 pttc = pht * 1.2 'any other value than' luxury '
End Select
'return the result
MyTTCAccording = pttc
End Function
```

It is possible to introduce value ranges in the part Case of the structure Select Case. The comparison becomes more sophisticated. Variable is a digital in this case, integer or even real.

Syntax

```
Select Case variable
 Case Is comparison op value
 instruction block

 Case starting value To end value
 instruction block

 Case Else
 instruction block
End Select
```

Multiple branching SELECT CASE - Value ranges - An example

Entry: quantity (integer number)
Output: unit price (real number)
Calculation: quantity <100 → u.p.= 0.5
100 ≤ quantity ≤ 200 → u.p.= 0.3
quantity> 200 → u.p. = 0.2

```
'calculation of the unit price according to the quantity
Public Function MyPU(quantity As Long) As Double
'intermediate variable
Dim pu As Double
'according to quantity values
Select Case quantity
 Case Is < 100
 pu = 0.5
 Case 100 To 200
 pu = 0.3
 Case Is > 200 'Case Else would have done the trick too
 pu = 0.2
End Select
MyPU = pu
End Function
```

Repeat the execution of a block of instructions. The number of iterations is controlled by an index.

Syntax

```
For index = val.departure To val.end Step value
 instruction block
 ...
Next index
```

(1) Index is an ordered type, very often a numeric

(2) value controls passing from one index value to another, if omitted, value = 1 by default

(3) Next confirms the change to the next value of index, if this next value is> to val.end, we get out of the loop

(4) val.end must be superior at val.departure so that we get into the loop

(5) If value is negative, val.end must be less than val.departure this time

(6) Instruction Exit For allows you to get out of the loop prematurely

(7) You can nest loops (a loop inside another loop)

Entry: n (integer number)
Output: S (real number)
Calculation: $S = 1^2 + 2^2 + \ldots + n^2$

```
'calculating the sum of the squares of the values
Public Function MySumCar(n As Long) As Double
'calculation variables (s for the sum, i: index)
Dim s As Double, i As Long
'initialization
s = 0
'loop with index i
For i = 1 To n Step 1
 s = s + i ^ 2
'Next plays the role of incrementation (following i)
Next i
'return the result
MySumCar = s
End Function
```

Repeat the execution of a block of instructions. The number of iterations is controlled by a condition. Watch out for the infinite loop i.e. the condition allowing to exit the loop is never triggered.

Syntax

```
Do While condition
 Instruction block ...
 ...
Loop
```

(1) Condition is a boolean, it is often a comparison operation

(2) We continue execution AS LONG as the condition is true; if the condition is false, we exit the loop

(3) Exit Do allows triggering the premature exit of the loop

If the condition is immediately false.

We cannot go into the loop.

Entry: n (integer number)
Output: S (real number)
Calculation: $S = 1^2 + 2^2 + \ldots + n^2$

```
'calculating the sum of the squares of the values
Public Function MyWhileSum(n As Long) As Double
'calculation variables
Dim s As Double, i As Long
'initialization
s = 0
'it is also up to us to initialize the index
i = 1
'loop WHILE
Do While (i <= n)
 'summon
 s = s + i ^ 2
 'no next, we need to increment the index
 i = i + 1
Loop
'return the result
MyWhileSum = s
End Function
```

Repeat the execution of a block of instructions. The number of iterations is controlled by a condition.

Syntax

```
Do
 Instruction block
 ...
 ...
Loop While condition
```

We are sure to enter the loop at least once.

Choosing the right structure (Do .. While or While .. Do) depends on the problem to be treated.

Condition-controlled « Do » loops are very rich in VBA.

```
Do{While | Until}condition
 [statements]
 [Exit Do]
 [statements]
Loop
-or-
Do
 [statements]
 [Exit Do]
 [statements]
Loop{While | Until}condition
```

Do Loop…Until also exists.

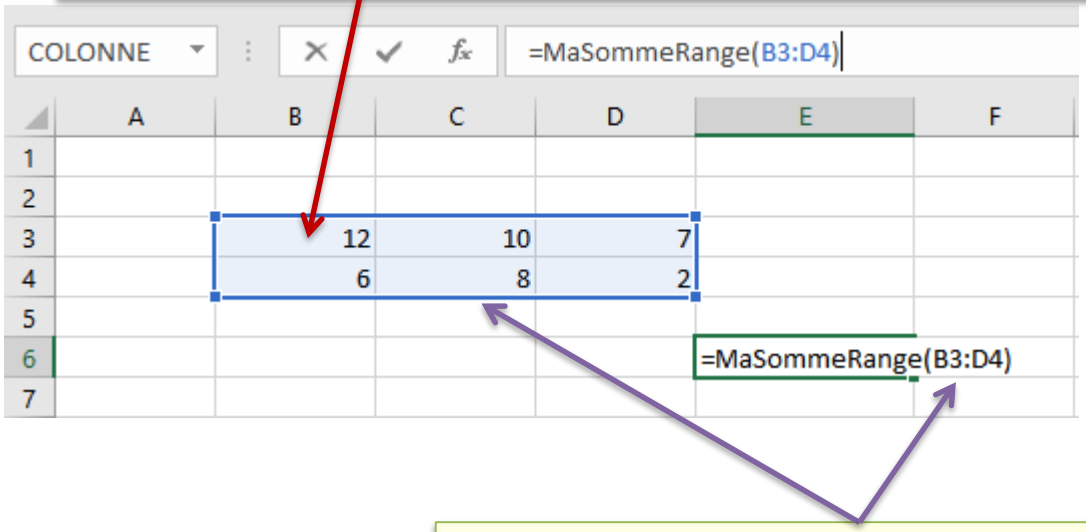
The Excel-specific "range of cells" type

THE RANGE TYPE

The RANGE type designates a cell range, this is an Excel specific type.

Example

Top and left corner of the range of cells passed in parameter of the function = coordinate (1, 1) i.e. row n ° 1 and column n ° 1,whatever the absolute position of the range in the worksheet (here the northwest corner is in B3)



A block of cells (B3: D4) is passed as a parameter of the function. This block is necessarily rectangular, with, here: 2 rows and 3 columns.

Function MySumRange() is supposed to do the same as the standard SUM () Excel function.

Entrance: plage (range)
Output: s (real number)
Calculation: Sum of values

Rows and columns start at index 1, regardless of the position of the range in the sheet.

```
'Working on the Range type
Public Function MySumRange(plage As Range) As Double
'intermediate variables
Dim s As Double, i As Long, j As Long
'initialization of the sum
s = 0
'walk through the cell range
For i = 1 To plage.Rows.Count Step 1 'lines
 For j = 1 To plage.Columns.Count Step 1 'columns
 'read values and sum
 s = s + plage.Cells(i,j).Value
 Next j
Next i
'return the result
MySumRange = s
End Function
```

Number of rows in the cell range.

Number of columns.

Access to the value of the cell: row n ° i, column n ° j

The loop For Each is adapted to the route of the collections.

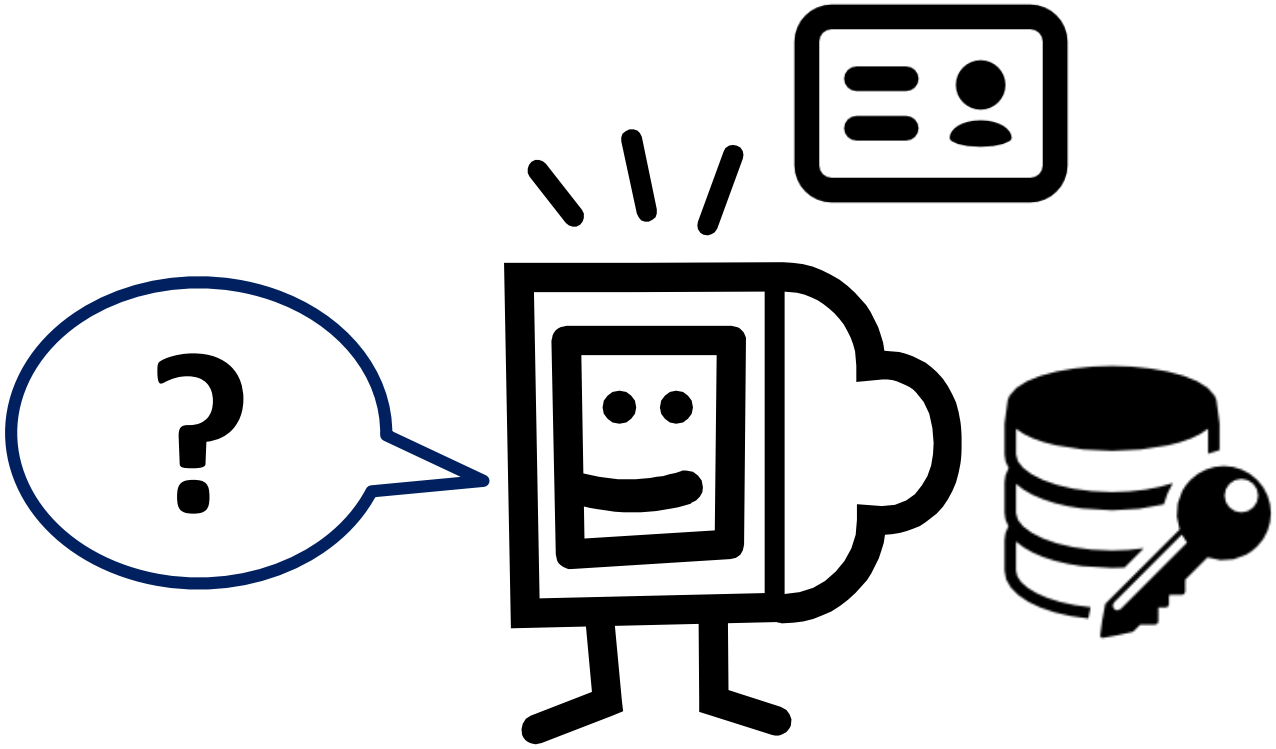
However, a range of cells is a collection of cells.

```
'Working on the Range type with a For Each
Public Function MySumRangeEach(plage As Range) As Double
'intermediate variables
Dim s As Double, cell As Range
'initialization of the sum
s = 0
'walk through the cell range
For Each cell In plage
 s = s + cell.Value
Next cell
'return the result
MySumRangeEach = s
End Function
```

A cell is a range of cells with a single cell.

plage is a collection to be treated. No matter the direction of the route here (row by row, or column by column).

It is indeed the value contained in the cell which is used for the sum.



To be continued with the Variant Type…

Thank you

Herve Hocquard(hocquard@labri.fr)

http://www.labri.fr/perso/hocquard/Teaching.html