Lesson 4

Sébastien Mathier

www.excel-pratique.com/en

Conditions :

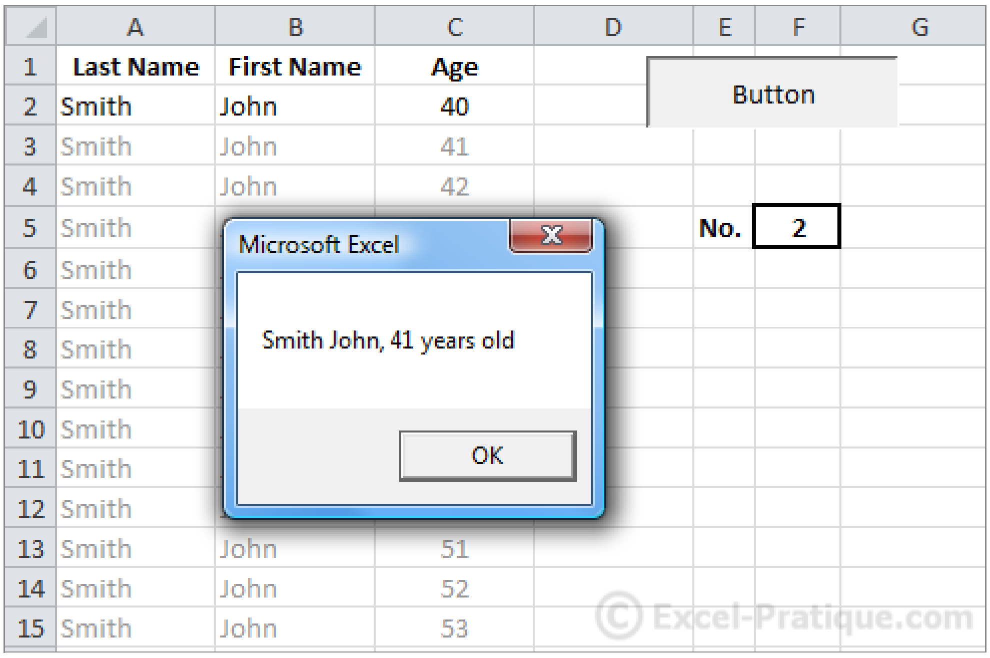
Conditions are very useful in programming because they allow us to execute actions based on specific criteria (it's the same principle as the IF function).

The most important conditional function is **If**, and now we'll take a look at how it works :

```
If [CONDITION HERE] Then ' => IF condition is validated, THEN
   'Instructions if true
Else ' => OTHERWISE
   'Instructions if false
End If
```

Let's move right on to practice and return to the example that we used in the lesson on variables. The purpose of this procedure was to display a dialog box containing the data from the row whose number is indicated in cell F5 :

Source file : **conditions.xls**



	A	B	C	D	E	F	G
1	Last Name	First Name	Age		Button		
2	Smith	John	40				
3	Smith	John	41				
4	Smith	John	42				
5	Smith				No.	2	
6	Smith						
7	Smith						
8	Smith						
9	Smith						
10	Smith						
11	Smith						
12	Smith						
13	Smith	John	51				
14	Smith	John	52				
15	Smith	John	53				

Microsoft Excel

Smith John, 41 years old

OK

If you enter a letter in cell F5, it will cause a bug. We want to prevent that from happening.

```
Sub variables()
   'Declaring variables
   Dim last_name As String, first_name As String, age As Integer, row_number As Integer

   'Assigning values to variables
   row_number = Range("F5") + 1
   last_name = Cells(row_number, 1)
   first_name = Cells(row_number, 2)
   age = Cells(row_number, 3)

   'Dialog box
   MsgBox last_name & " " & first_name & ", " & age & " years old"
End Sub
```

Let's begin by adding a condition that will verify that the value of cell F5 is numerical before the code is executed.

We'll use the function **IsNumeric** to test this condition :

```
Sub variables()

  'If the value in parentheses (cell F5) is numerical (AND THEREFORE IF THE CONDITION IS TRUE) then
  'execute the instructions that follow THEN
   If IsNumeric(Range("F5")) Then

       'Declaring variables
       Dim last_name As String, first_name As String, age As Integer, row_number As Integer
       'Values of variables
       row_number = Range("F5") + 1
       last_name = Cells(row_number, 1)
       first_name = Cells(row_number, 2)
       age = Cells(row_number, 3)
       'Dialog Box
       MsgBox last_name & " " & first_name & ", " & age & " years old"

    End If

End Sub
```

We also need to add instructions to execute in case the conditions are not met :

```
Sub variables()

    If IsNumeric(Range("F5")) Then 'IF CONDITION TRUE

       'Declaring variables
       Dim last_name As String, first_name As String, age As Integer, row_number As Integer
       'Values of variables
       row_number = Range("F5") + 1
       last_name = Cells(row_number, 1)
       first_name = Cells(row_number, 2)
       age = Cells(row_number, 3)
       'Dialog box
       MsgBox last_name & " " & first_name & ", " & age & " years old"

    Else 'IF CONDITION FALSE

       'Dialog box : warning
       MsgBox "Your entry" & Range("F5") & " is not valid !"
       'Deleting the contents of cell F5
       Range("F5").ClearContents

    End If

End Sub
```

Now non-numerical values won't cause any problems.

Working with our array, which contains 16 rows of data, our next step will be to test whether the variable **row_number** soit : "greater than or equal to 2" and "less than or equal to 17".

But first, have a look at this list of comparison operators :

=	is equal to
<>	is different than
<	is less than
<=	is less than or eual to
>	is greater than
>=	is greater than or equal to

And these other useful operators :

AND	[condition1] AND [condition2] The two conditions must be true
OR	[condition1] OR [condition2] At least 1 of the 2 conditions must be true
NOT	NOT [condition1] The condition should be false

Now let's add the conditions that we mentioned above, using **AND** along with the comparison operators listed above :

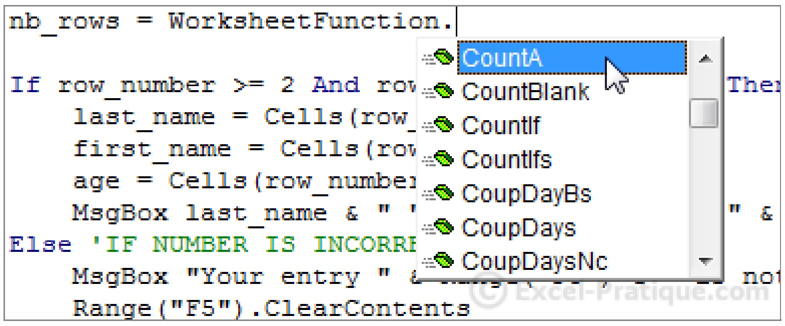
```vba
Sub variables()
    If IsNumeric(Range("F5")) Then 'IF NUMERICAL
        Dim last_name As String, first_name As String, age As Integer, row_number As Integer
        row_number = Range("F5") + 1

        If row_number >= 2 And row_number <= 17 Then 'IF CORRECT NUMBER
            last_name = Cells(row_number, 1)
            first_name = Cells(row_number, 2)
            age = Cells(row_number, 3)
            MsgBox last_name & " " & first_name & ", " & age & " years old"
        Else 'IF NUMBER IS INCORRECT
            MsgBox "Your entry " & Range("F5") & " is not a valid number !"
            Range("F5").ClearContents
        End If

    Else 'IF NOT NUMERICAL
        MsgBox "Your entry " & Range("F5") & " is not valid !"
        Range("F5").ClearContents
    End If
End Sub
```

If we wanted to make our macro a bit more practical, we could replace **17** with a variable that contained the number of rows. This would let us add or remove lines from our array without having to change this limit each time.

In order to do this, we have to create a variable **nb_rows** and add this function:



```vba
nb_rows = WorksheetFunction.

If row_number >= 2 And row                    Then
    last_name = Cells(row_
    first_name = Cells(row
    age = Cells(row_numbe
    MsgBox last_name & " "                    " &
Else 'IF NUMBER IS INCORRE
    MsgBox "Your entry " &                    not
    Range("F5").ClearContents
```

In this case, we'll use **WorksheetFunction.CountA** which is the **function COUNTA** which you may already be familiar with ...

We want this function to count the number of non-empty cells in the first column and then replace **17** with **nb_rows** :

```
Sub variables()
    If IsNumeric(Range("F5")) Then 'IF NUMERICAL
        Dim last_name As String, first_name As String, age As Integer, row_number As Integer
        Dim nb_rows As Integer

        row_number = Range("F5") + 1
        nb_rows = WorksheetFunction.CountA(Range("A:A")) 'NBVAL Function

        If row_number >= 2 And row_number <= nb_rows Then 'IF CORRECT NUMBER
            last_name = Cells(row_number, 1)
            first_name = Cells(row_number, 2)
            age = Cells(row_number, 3)
            MsgBox last_name & " " & first_name & ", " & age & " years old"
        Else 'IF NUMBER IS INCORRECT
            MsgBox "Your entry " & Range("F5") & " is not a valid number !"
            Range("F5").ClearContents
        End If

    Else 'IF NOT NUMERICAL
        MsgBox "Your entry " & Range("F5") & " is not valid !"
        Range("F5").ClearContents
    End If
End Sub
```

ElseIf :

ElseIf makes it possible to add more conditions after the IF command :

```
If [CONDITION 1] Then ' => IF condition 1 is true, THEN
   'Instructions 1
ElseIf [CONDITION 2] Then ' => IF condition 1 is false, but condition 2 is true, THEN
   'Instructions 2
Else ' => OTHERWISE
   'Instructions 3
End If
```

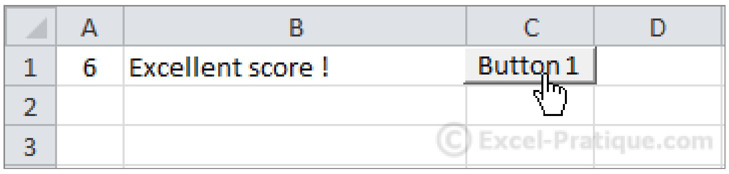
If condition 1 is true, Instructions 1 will be executed, and we will leave the **If** command (which begins with **If** and ends with **End If**). If condition 1 is false, we continue to condition 2. If this condition is true, Instructions 2 will be executed, and if it is false, then Instructions 3 (under Else) will be executed.

Here is an example, with a score between 1 and 6 in cell A1 (without decimals in this case) and a score_comment in cell B1 based on the score:

```
Sub scores_comment()
    'Variables
    Dim note As Integer, score_comment As String
    note = Range("A1")

    'Comments based on the score
    If note = 6 Then
        score_comment = "Excellent score !"
    ElseIf note = 5 Then
        score_comment = "Good score"
    ElseIf note = 4 Then
        score_comment = "Satisfactory score"
    ElseIf note = 3 Then
        score_comment = "Unsatisfactory score"
    ElseIf note = 2 Then
        score_comment = "Bad score"
    ElseIf note = 1 Then
        score_comment = "Terrible score"
    Else
        score_comment = "Zero score"
    End If

    'Comments in B1
    Range("B1") = score_comment
End Sub
```



	A	B	C	D
1	6	Excellent score !	Button 1	
2				
3				

Select :

There is an alternative to using **If** with lots of **ElseIf** instructions : the **Select** command, which is better suited to these sorts of situations.

Here is an example of the same macro written with **Select** :

```
Sub scores_comment()
    'Variables
    Dim note As Integer, score_comment As String
    note = Range("A1")

    'Comments based on the score
    Select Case note     ' <= the value to test (the score, in this case)
    Case Is = 6          ' <= if the value = 6
       score_comment = "Excellent score !"
    Case Is = 5          ' <= if the value = 5
       score_comment = "Good score"
    Case Is = 4          ' <= if the value = 4
       score_comment = "Satisfactory score"
    Case Is = 3          ' <= if the value = 3
       score_comment = "Unsatisfactory score"
    Case Is = 2          ' <= if the value = 2
       score_comment = "Bad score"
    Case Is = 1          ' <= if the value = 1
       score_comment = "Terrible score"
    Case Else            ' <= if the value isn't equal to any of the above values
       score_comment = "Zero score"
    End Select

    'Comment in B1
    Range("B1") = score_comment
End Sub
```

Please note that we could also have used other comparison operators, for example :

```
Case Is >= 6         'if the value is >= 6
```

Examples with different values :

```
Case Is = 6, 7       'if the values is = 6 or 7
Case Is <> 6, 7      'if the value is different than 6 or 7
```

```
Case 6 To 10         'if the value is = any number between 6 and 10
```

A conditional based on type :

IsNumeric (this is the function we used on the last page) returns TRUE if the value is numerical and FALSE if this is not the case :

```
If IsNumeric(Range("A1")) = True Then 'IF THE VALUE IS NUMERICAL ...
```

The following code has the same effect as the version above (we don't have to include the **= True** because it is the default to ask whether a condition is true) :

```
If IsNumeric(Range("A1")) Then 'IF THE VALUE IS NUMERICAL ...
```

If what we wanted to do was to test whether the value was NOT numerical, we could do this in either of the following two ways :

```
If IsNumeric(Range("A1")) = False Then 'IF THE VALUE IS NOT NUMERICAL...
```

```
If Not IsNumeric(Range("A1")) Then 'IF THE VALUE IS NOT NUMERICAL ...
```

Here are some examples of other functions similar to **IsNumeric** :

```
If IsDate(Range("A1")) Then 'IF THE VALUE IS A DATE ...
```

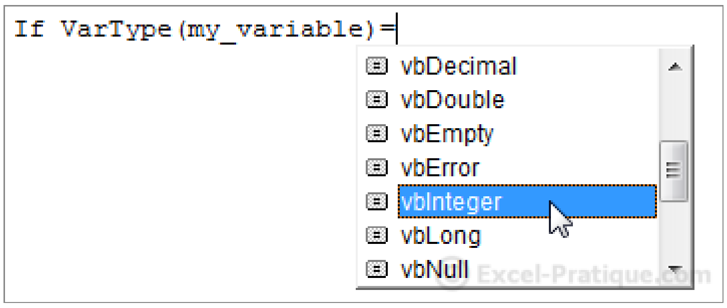
```
If IsEmpty(Range("A1")) Then 'IF EMPTY...
```

```
If var_object Is Nothing Then 'IF OBJECT IS NOT SET ...
```

A conditional based on a variable's type :

To execute commands based on the type of a variable (Variant), we will need to use the **VarType** function.

The list of types will appear once we have entered the = operator :



```
If VarType(my_variable)=
```

```
If VarType(my_variable) = vbInteger Then 'IF my_variable if is of type Integer ...
```

Values of constants:

Constant	Value
vbEmpty	0
vbNull	1
vbInteger	2
vbLong	3
vbSingle	4
vbDouble	5
vbCurrency	6
vbDate	7
vbString	8
vbObject	9
vbError	10

```
If VarType(my_variable) = vbInteger Then 'IF my_variable is of type Integer ...
'Identical to :
If VarType(my_variable) = 2 Then 'IF my_variable is of type Integer ...
```

A conditional based on comparing two character strings :

Up to this point, we have seen this :

```
my_variable = "Example 12345"

If my_variable = "Example 12345" Then ' => TRUE
```

In this case, the two strings are identical, it's nothing out of the ordinary ...

But if we want to test whether the variable contains the value "12345" without taking any of the other characters into account, we would have to use the **Like** command and a * operator before and after the search value.

The * operator stands in for : any character or multiple characters :

```
my_variable = "Example 12345"

If my_variable Like "*12345*" Then ' => TRUE
```

The **#** operator can stand in for any numerical character from 0 to 9 :

```
my_variable = "Example 12345"

If my_variable Like "Example 12###" Then ' => TRUE
```

The **?** operator can stand in for any single character :

```
my_variable = "Example 12345"

If my_variable Like "?xample?1234?" Then ' => TRUE
```

We can also use a particular character or range of characters in the same way :

- **[abc]** stands in for any one of the following characters : a b c
- **[a-g]** stands in for any one of the following characters : a b c d e f g
- **[369]** stands in for any one of the following characters : 3 6 9
- **[2-5]** stands in for any one of the following characters : 2 3 4 5
- **[?*#]** stands in for any one of the following characters : ? * #

```
my_variable = "Example 12345"

If my_variable Like "[DEF]xample 1234[4-7]" Then ' => TRUE
```

An **!** added after the **[** will stand in for any character not included within the bracketed expression :

```
my_variable = "Example 12345"

If my_variable Like "[!GHIJ]xample 1234[!6-9]" Then ' => TRUE
```

Comment : an uppercase character is not equivalent to the same character in lowercase in this context. If you don't want to make distinctions between upper and lowercase characters, put an **Option Compare Text** command at the beginning of the module.