Excel
VBA programming
Visual Basic for Applications

Herve Hocquard

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

INTRODUCTION

Algorithm

- "IT" solution relating to a problem
- Sequence of actions (instructions) applied on data
- 3 main stages:
    1. data entry (reception)
    2. treatments
    3. restitution (application) of results

Program

- Transcription of an algorithm with a predefined syntax
- **Visual Basic for Applications**
- Same basic principles as other object languages (Java, C #, etc.)
- VBA interacts with the predefined functions available in the Office suite

Visual Basic for Applications

Many hard-core programmers scoff at the idea of programming in BASIC. The name itself (an acronym for Beginner's All-purpose Symbolic Instruction Code) suggests that BASIC isn't a professional language.

In fact, BASIC was first developed in the early 1960s as a way to teach programming techniques to college students. BASIC caught on quickly and is available in hundreds of dialects for many types of computers.

BASIC gained quite a bit of respectability in 1991 when Microsoft released Visual Basic for Windows.

This product made it easy for the masses to develop stand-alone applications for Windows.

Visual Basic has very little in common with early versions of BASIC, but Visual Basic is the foundation on which VBA was built.

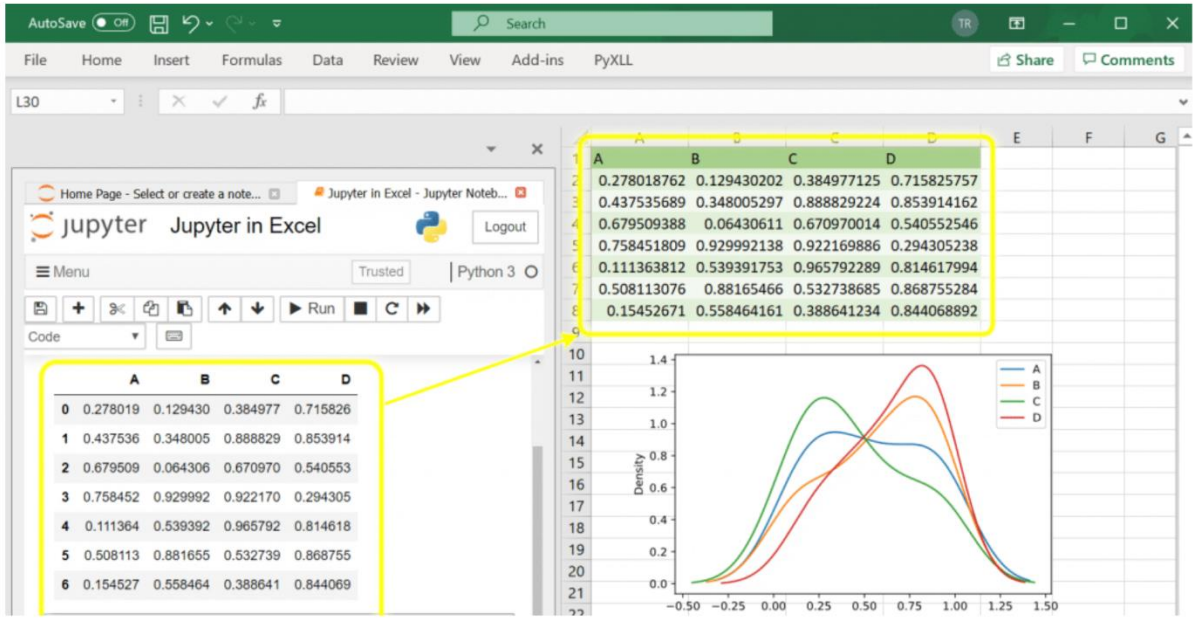
The secret to using VBA with other applications lies in understanding the *object model* for each application.
VBA, after all, simply manipulates objects, and each product (Excel, Word, Access, PowerPoint, and so on) has its own unique object model. You can program an application by using the objects that the application exposes.

Excel's object model, for example, exposes several powerful data analysis objects, such as worksheets, charts, pivot tables, and numerous mathematical, financial, engineering, and general business functions. With VBA, you can work with these objects and develop automated procedures. While you work with VBA in Excel, you gradually build an understanding of the object model.

Warning: The object model will be confusing at first. Eventually, however, the pieces come together and all of a sudden, you realize that you've mastered it!

For the past few years, I've heard rumors that Microsoft is going to remove VBA from the Office applications and replace it with .NET (or something else). My understanding is that these rumors are unfounded. Sure, Microsoft has developed another way to automate Office applications, but VBA will be around for quite a while — at least in Excel for Windows.

Microsoft did remove VBA from Excel for Mac, but then they put it back in!



Jupyter Notebooks in Microsoft Excel. Image by the author.

Why will VBA survive? Because millions of VBA-based solutions are in use and VBA is much easier to learn and use than the alternatives.

Following is a quick-and-dirty summary of what VBA is all about:

- **Code**: You perform actions in VBA by executing VBA code. You write (or record) VBA code, which is stored in a VBA module.

- **Module:** VBA modules are stored in an Excel workbook file, but you view or edit a module by using Visual Basic Editor (VBE). A VBA module consists of procedures.

- **Procedures:** A procedure is basically a unit of computer code that performs some action.

VBA supports two types of procedures: Sub procedures and Function procedures.

● **Sub:** A Sub procedure consists of a series of statements and can be executed in a number of ways. Here's an example of a simple Sub procedure called Test: This procedure calculates a simple sum and then displays the result in a message box.

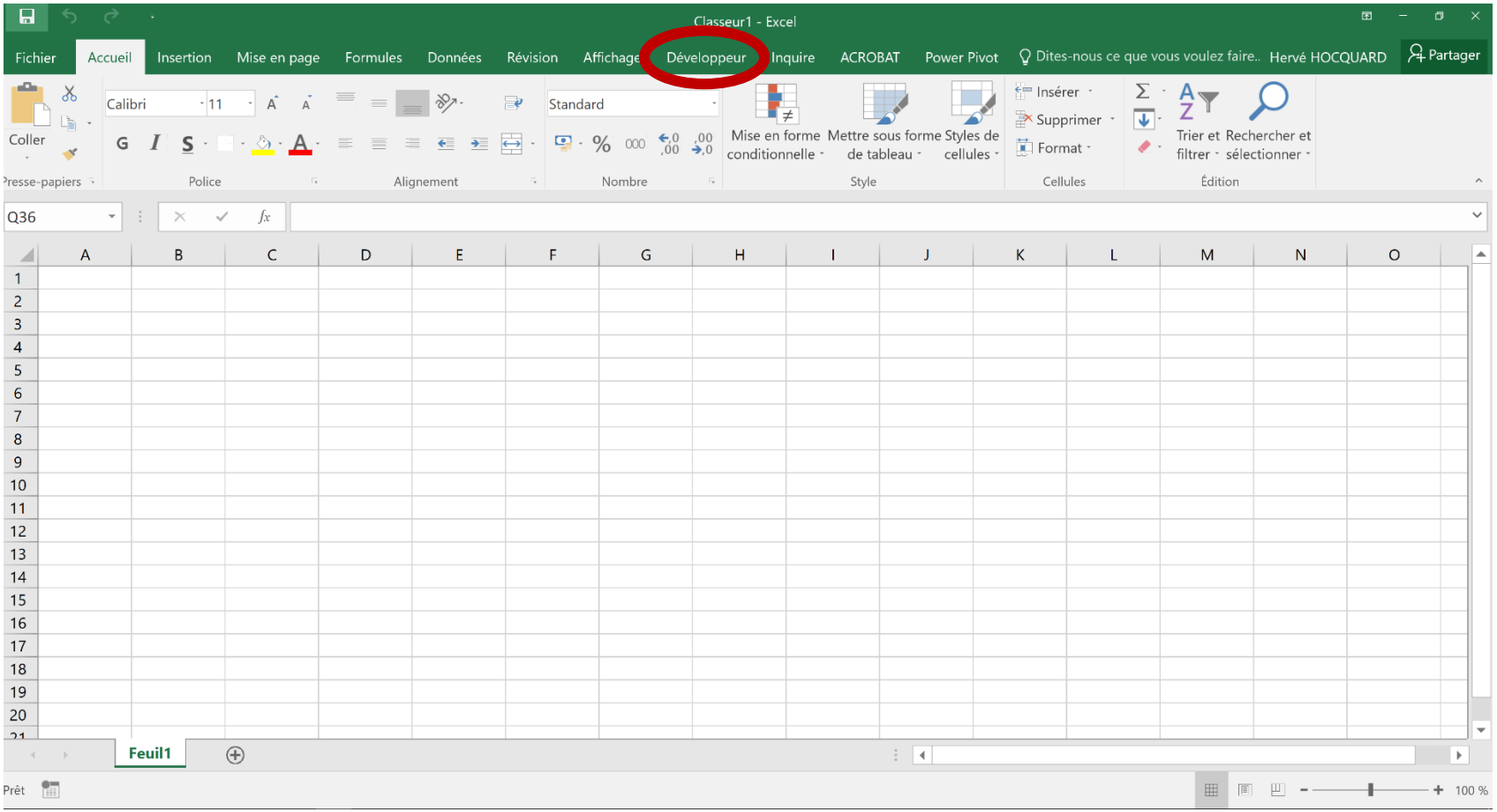
```
Sub Test()
    sum = 1 + 1
    MsgBox "The answer is " & sum
End Sub
```

● **Function:** A Function procedure returns a single value (or possibly an array). A Function can be called from another VBA procedure or used in a worksheet formula. Here's an example of a Function named AddTwo:

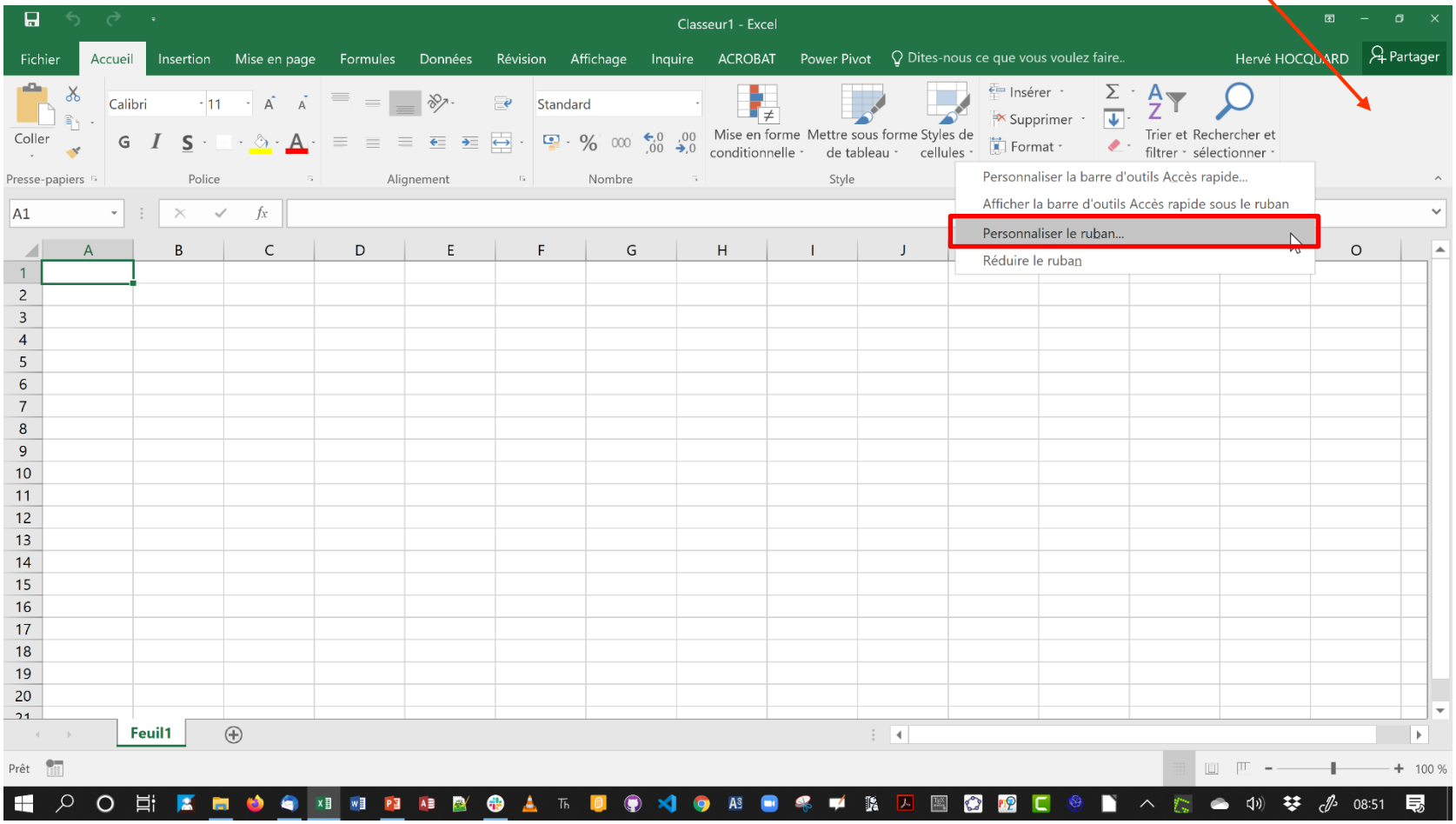
```
Function AddTwo(arg1, arg2)
    AddTwo = arg1 + arg2
End Function
```

THE VISUAL BASIC EDITOR-VBE

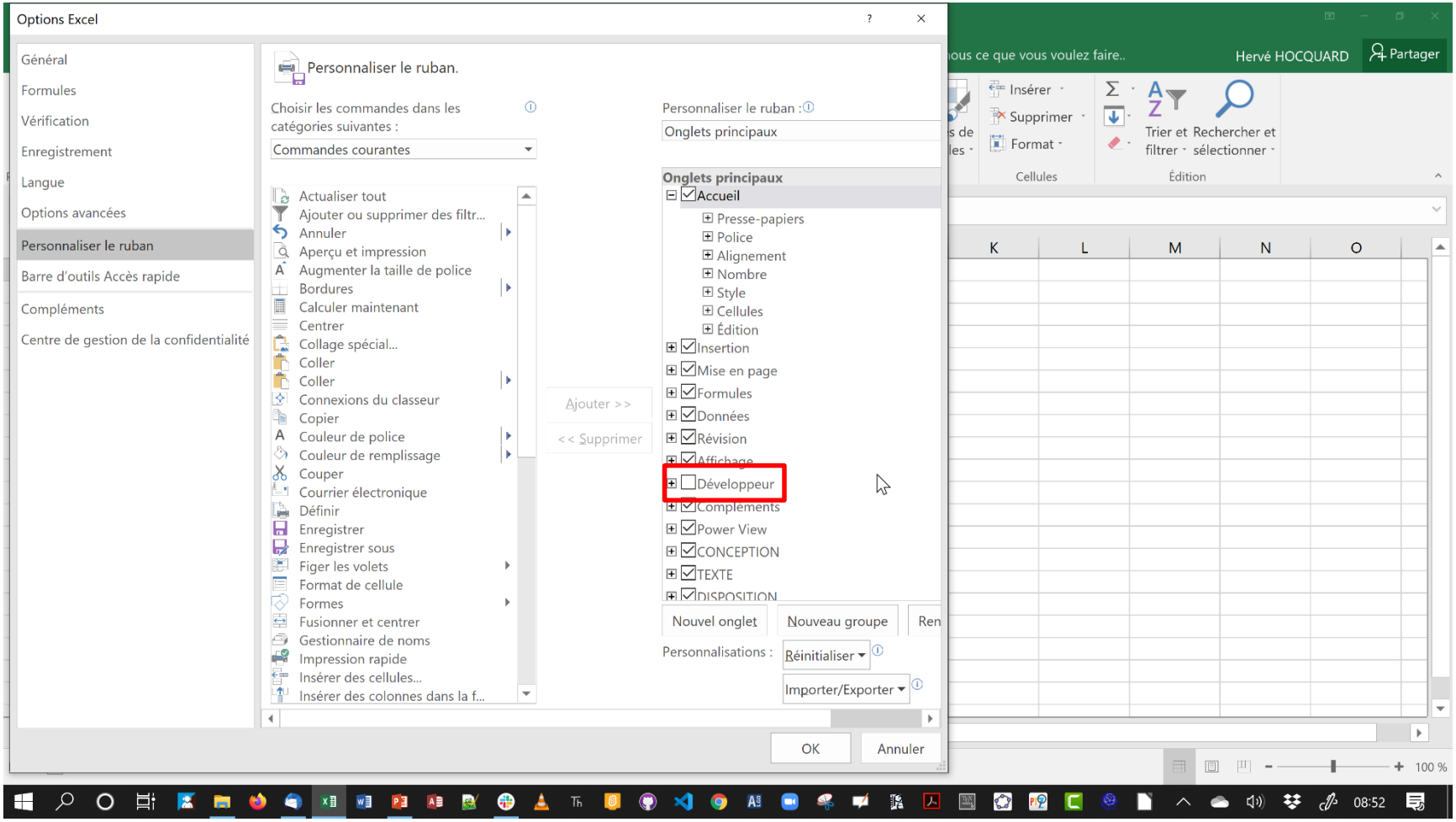
The Editor (Visual Basic Editor)



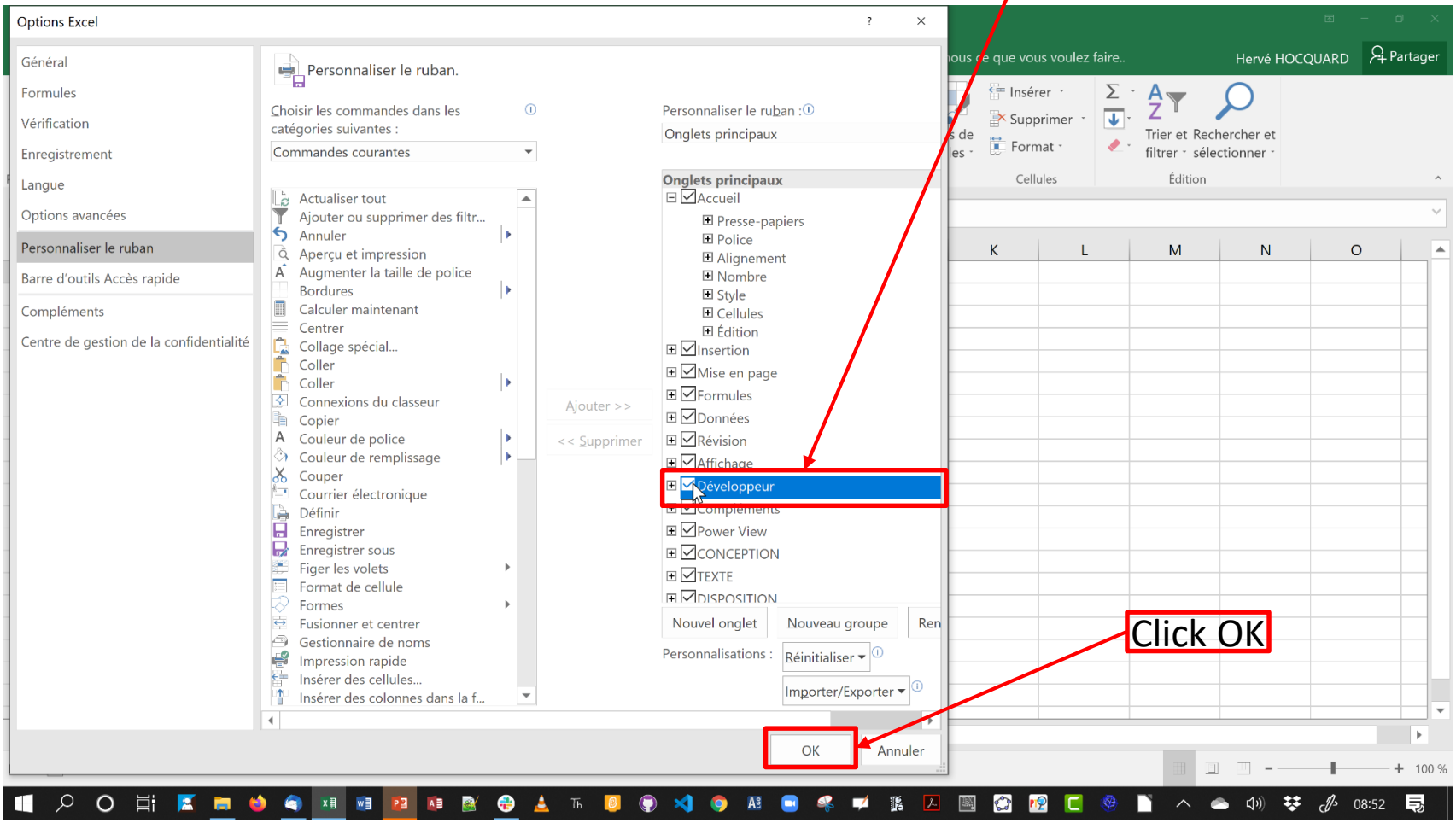
Right-click anywhere on the Ribbon and choose Personnaliser le ruban…



Excel displays the “Personnaliser le ruban” tab of the Excel Options dialog box

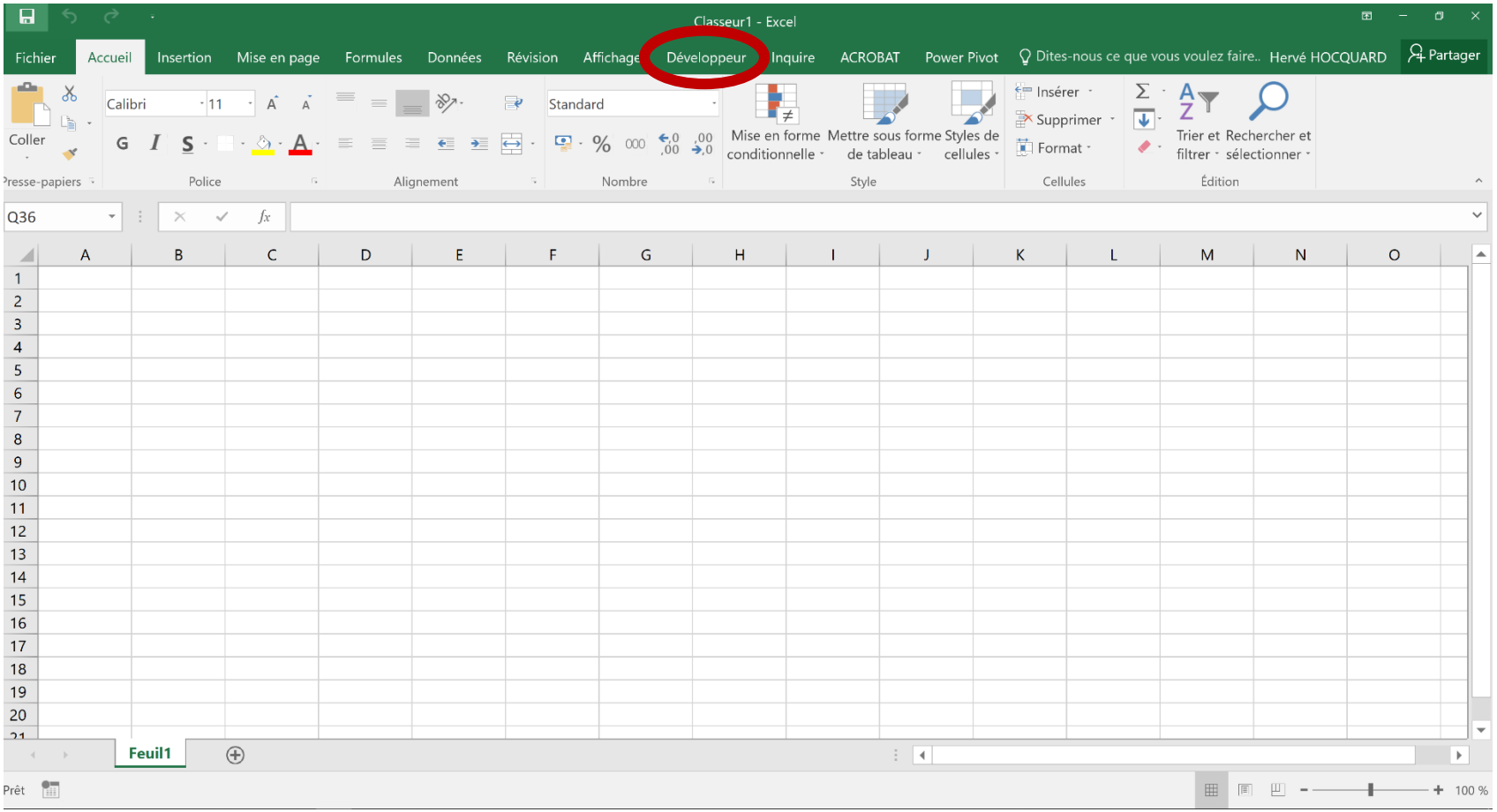


In the list box on the right, place a check mark next to Développeur

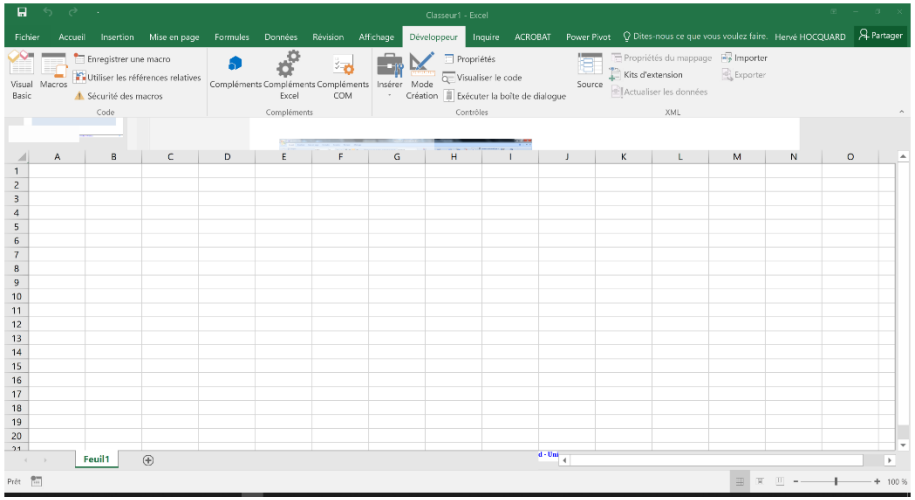


Click OK

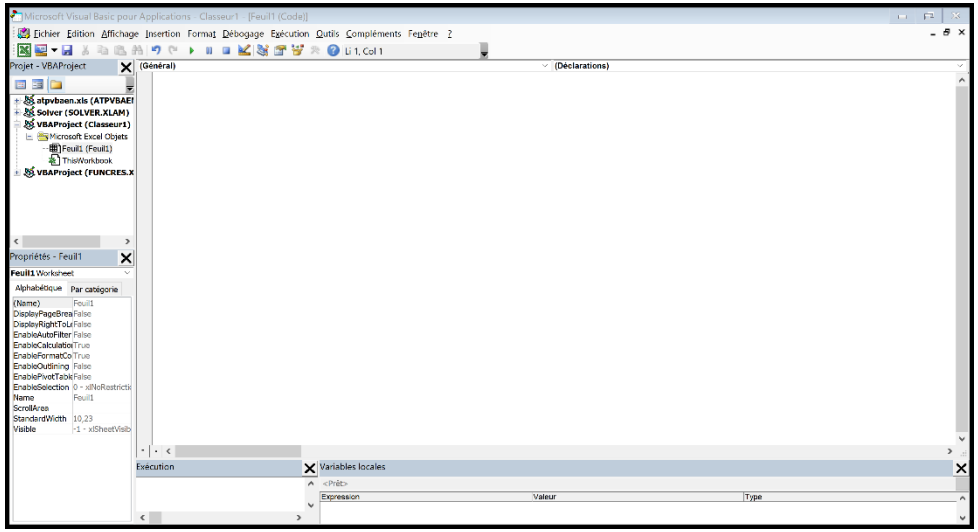
The Editor (Visual Basic Editor)



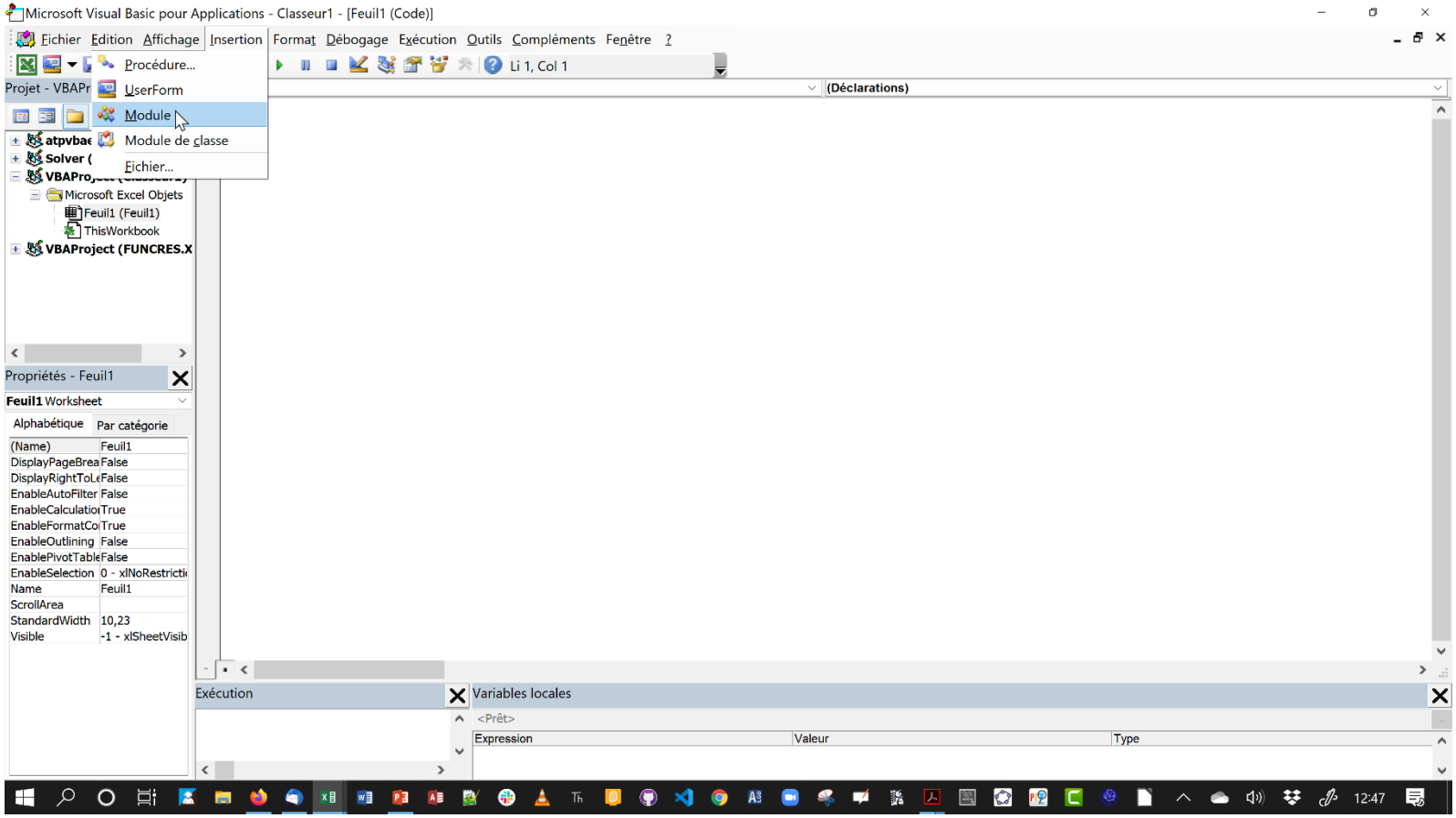
The Editor (Visual Basic Editor)



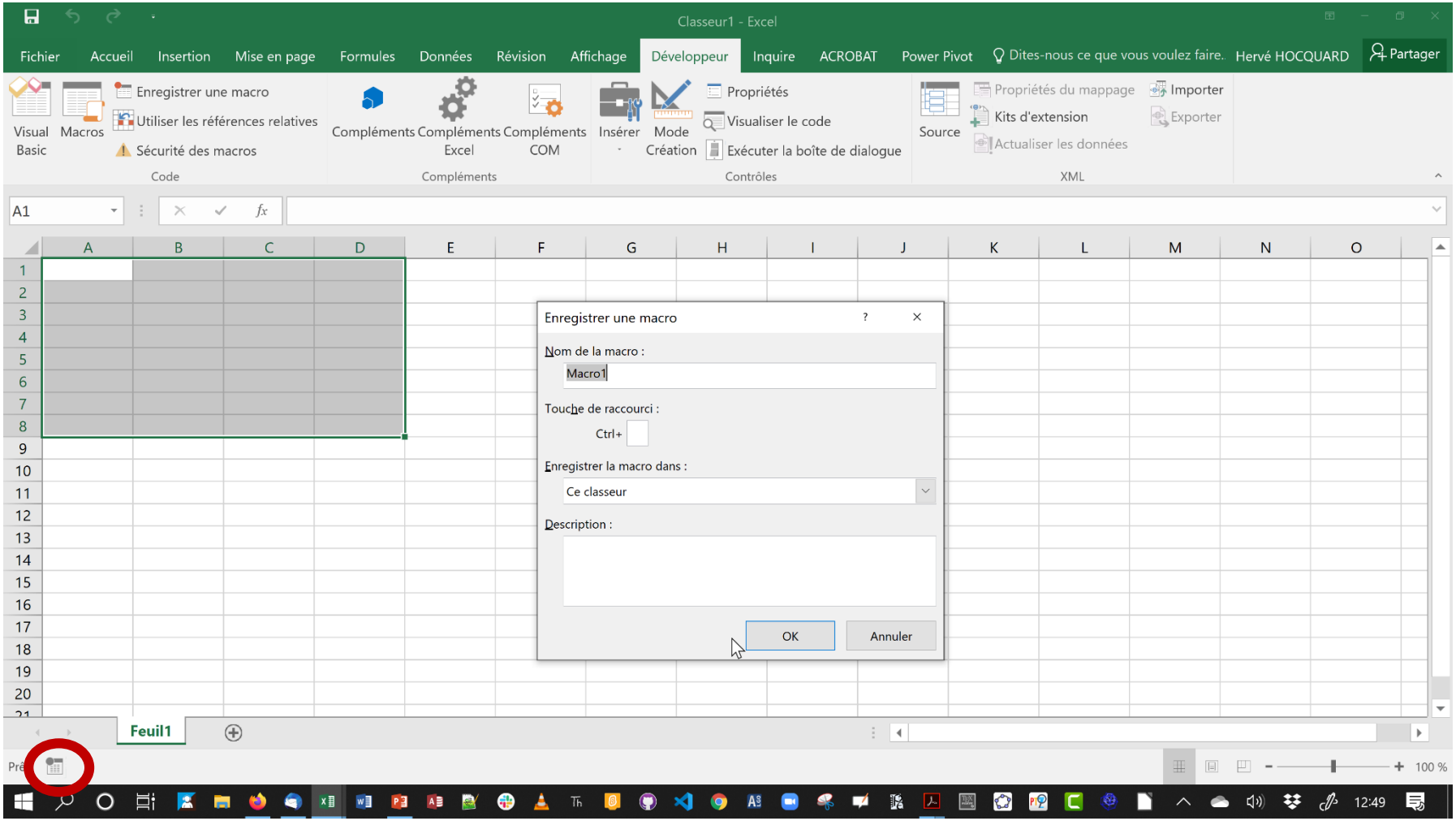
Alt + F11



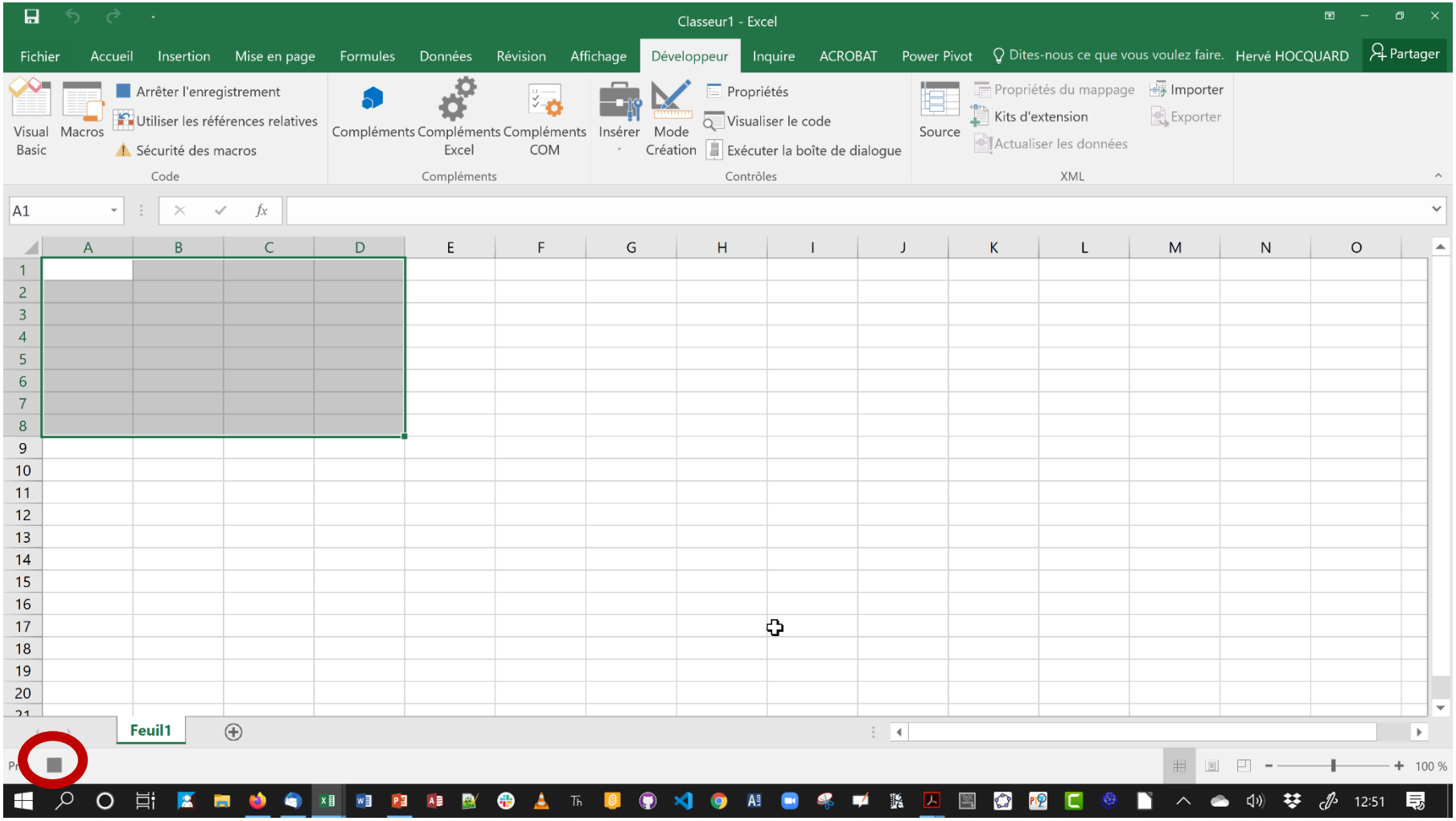
Inserting a module into the editor



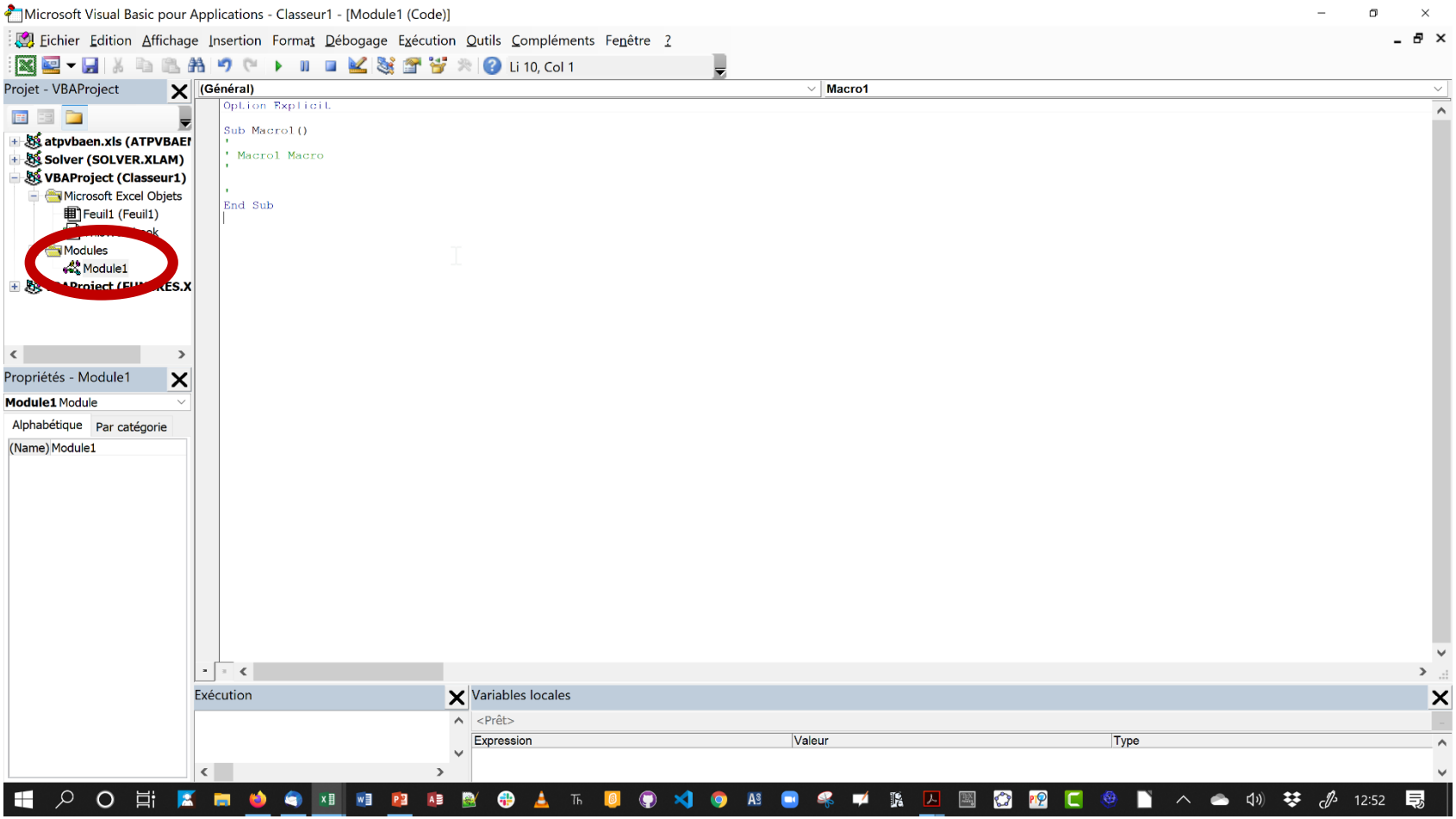
Record a macro



Record a macro



Impact in the editor

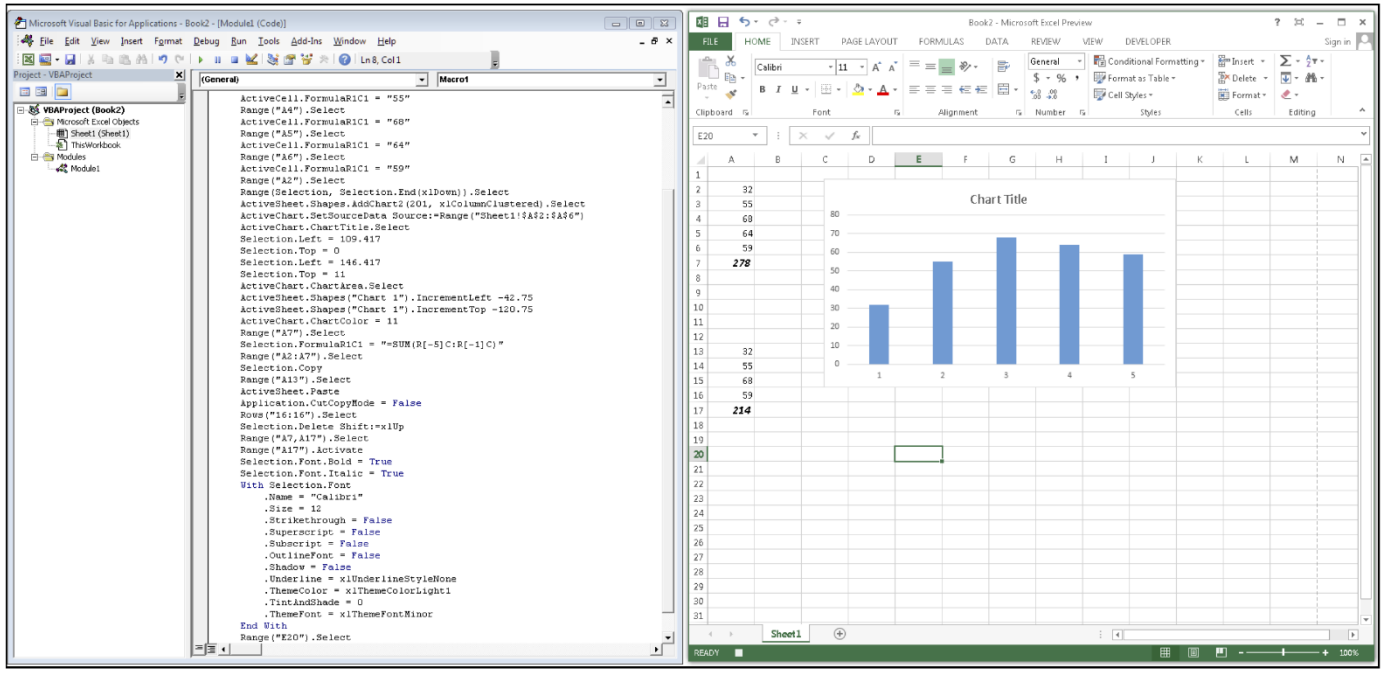


The Excel macro recorder translates your mouse and keyboard actions into VBA code. I could probably write several pages describing how this translation occurs, but the best way to show you is by example. Follow these steps:

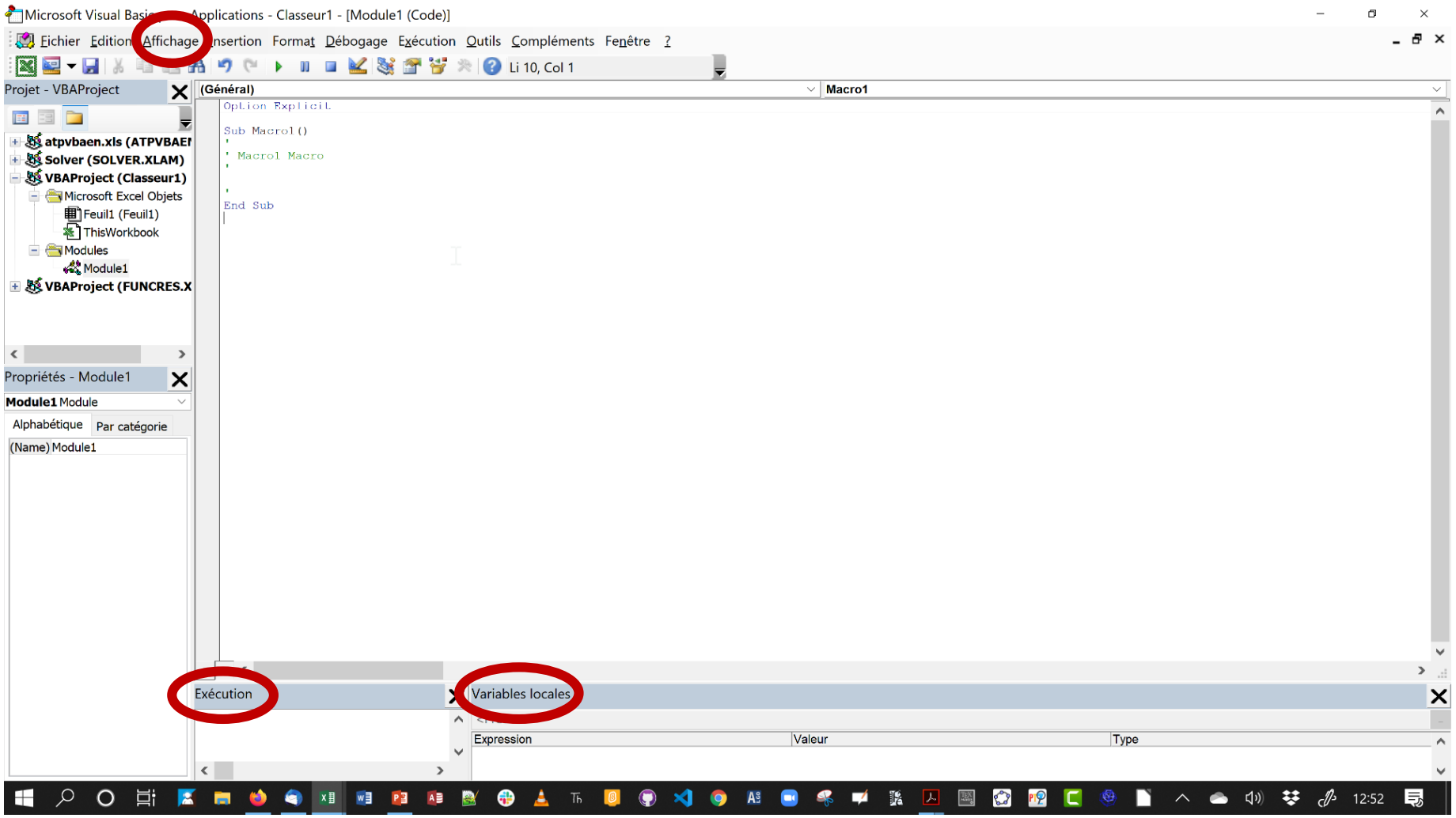
1. Start with a blank workbook.
2. Make sure that the Excel window isn't maximized.
You don't want it to fill the entire screen.
3. Press Alt+F11 to activate the VBE window.
Note: Make sure that this window isn't maximized. Otherwise, you won't be able to see the VBE window and Excel's window at the same time.
4. Resize and arrange Excel's window and the VBE window so that both are visible. (For best results, minimize any other applications that are running.)
5. Activate Excel, choose Développeur→Enregistrer une macro (Code), and then click OK to start the macro recorder.
6. Activate the VBE window.
7. In the Project Explorer window, double-click Module1 to display that module in the code window.
8. Close the Project Explorer window in VBE to maximize the view of the code window.

Your screen layout should look something like the example below. The size of the windows depends on your video resolution. If you happen to have a dual-display system, just put the VBA window on one display and the Excel window on the other display.

Now move around in the worksheet and select various Excel commands. Watch while the code is generated in the window that displays the VBA module. Select cells, enter data, format cells, use the Ribbon commands, create a chart, manipulate graphic objects, and so on. I guarantee that you'll be enlightened while you watch the code being spit out before your very eyes.



Customizing the VBE Environment



Throughout this course, I present many small snippets of VBA code to make a point or to provide an example. In some cases, this code consists of a single statement or only an expression, which isn't a valid instruction by itself.

For example, the following is an expression: `Range(''A1'').Value`

To test an expression, you must evaluate it. The MsgBox function is a handy tool for this: `MsgBox Range (''A1'').Value`

To try out these examples, put the statement in a procedure in a VBA module, like this:

```
Sub Test()
    'statement goes here
End Sub
```

Then put the cursor anywhere in the procedure and press F5 to execute it. Also, make sure that the code is being executed in the proper context. For example, if a statement refers to Sheet1, make sure that the active workbook has a sheet named Sheet1.

If the code is just a single statement, you can use the VBE “Exécution” window. The window “Exécution” is useful for executing a statement immediately — without having to create a procedure.

If the window “Exécution” isn’t displayed, press Ctrl+G in VBE.

Just type the VBA statement in the window “Exécution” and press Enter. To evaluate an expression in the window “Exécution”, precede the expression with a question mark (?), which is a shortcut for Print.

For example, you can type the following in the window “Exécution”:

```
? Range (''A1'').Value
```

The result of this expression is displayed in the next line of the “Exécution” window.

If you’d rather not deal with a series of message boxes, use this procedure to print the comments to the window “Exécution” in VBE:

```
Debug.Print Range (''A1'').Value
```

Before you can do anything meaningful, you must have some VBA code in a code window. This VBA code must be within a procedure. A procedure consists of VBA statements. For now, I focus on one type of code window: a VBA module.

You can add code to a VBA module in three ways:

- Enter the code manually. Use your keyboard to type your code.
- Copy and paste. Copy the code from another module (or from a website) and paste it into the module that you're working in.
- Use the macro-recorder feature. Use Excel's macro-recorder feature to record your actions and convert them into VBA code.

Sometimes, the most direct route is the best one. Entering code directly involves . . . well, entering the code directly. In other words, you type the code by using your keyboard. You can use the Tab key to indent the lines that logically belong together — for example, the conditional statements between the If and End If statements. Indenting isn't necessary, but it makes the code easier to read, so it's a good habit to acquire.
Entering and editing text in a VBA module works just as you would expect. You can select text, copy it or cut it, and then paste it to another location.

To get a feel for entering a VBA procedure, try this: Insert a VBA module into a project and then enter the following procedure in the code window of the module:

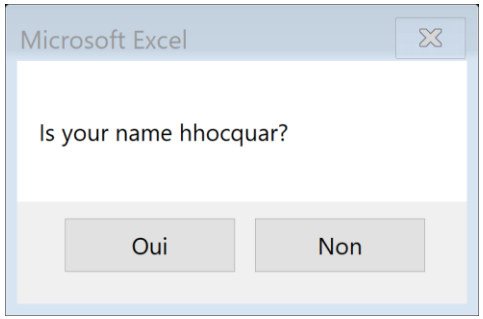
```
Sub SayHello()
     Msg = "Is your name " & Application.UserName & "?"
     Ans = MsgBox(Msg, vbYesNo)
     If Ans = vbNo Then
          MsgBox "Oh, never mind."
     Else
          MsgBox "I must be clairvoyant!"
     End If
End Sub
```

While you enter the code, note that VBE makes some adjustments to your text. For example, if you omit the space before or after an equal sign (=), VBE inserts the space for you. Also, the color of some of the text is changed. These adjustments are all perfectly normal, and you'll appreciate them later.

To execute the SayHello procedure, make sure that the cursor is located anywhere within the text that you typed. Then do any of the following:

- Press F5.
- Choose Run→Run Sub/UserForm.
- Click the Run Sub/UserForm button on the Standard toolbar.

If you entered the code correctly, the procedure executes, and you can respond to a simple dialog box that displays the username, as listed in the Excel Options dialog box. Notice that Excel is activated when the macro executes. At this point, it's not important that you understand how the code works; that becomes clear later in this course.



What you did in this exercise was write a VBA Sub procedure (also known as a macro). When you issued the command to execute the macro, VBE quickly compiled the code and executed it.
In other words, each instruction was evaluated, and Excel simply did what it was told to do. You can execute this macro any number of times (although it tends to lose its appeal after a while).
For the record, this simple procedure uses the following concepts (all of which I cover later in the course):

- Declaring a procedure (the first line)
- Assigning a value to variables (Msg and Ans)
- Concatenating strings (using the & operator)
- Using a built-in VBA function (MsgBox)
- Using built-in VBA constants (vbYesNo and vbNo)
- Using an If-Then-Else construct
- Ending a procedure (the last line)

Don’t forget to record the worbook with xlsm extension…

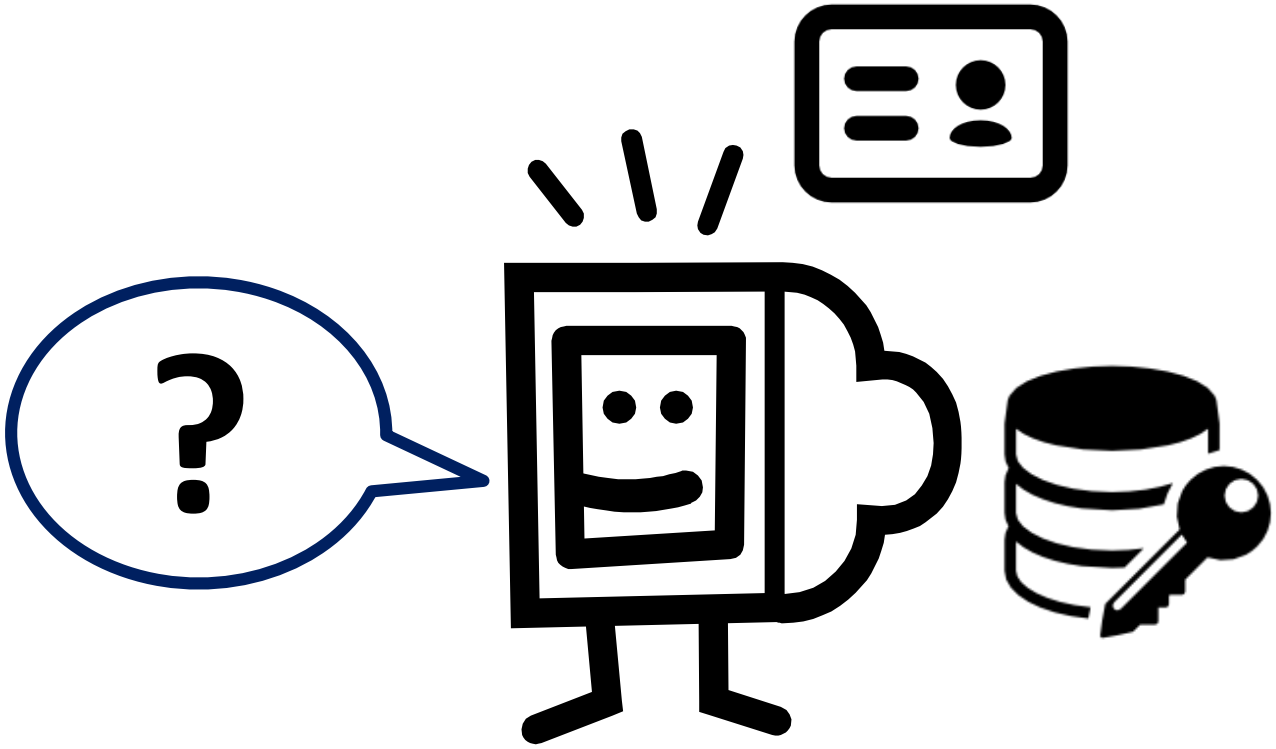
When you write VBA, or any programming language, you are going to encounter errors in it, or should we call them unintended features? Basically you can't write any substantial amount of code without needing to fix errors and make sure that it works as it's supposed to. We are going to look at the tools I use the most in the VBA editor to help debugging VBA code.

Stepping Through Code : F8
With the cursor in the sub, press **F8** to execute one line of code at a time. The next line to be executed will be highlighted in yellow, with a yellow arrow pointing to it. You can observe the variables and values in the windows “Exécution” and “Variables locales”.

Stepping Over Code : SHIFT + F8

Stepping Out of Code : CTRL + SHIFT + F8

Breakpoints : F9



To be continued with the object model in VBA…

Thank you

Herve Hocquard(hocquard@labri.fr)

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