Recitation 4

Name and NetID :

Section :

In this recitation we will explore some basic funtionality in Excel using Visual Basic for Applications (VBA).

1 Recording Macros

Open Excel. We will make a very simple calendar with the days of the week excluding the weekend. We want our first column to be M (for Monday), then T (for Tuesday), W, Th, F, repeat over and over. You can do this by dragging, but let's see if we can do this using a *macro*, typing the days of the week only once, and then repeating this by just executing the macro.

We want to click on the `Developer` tab at the top of Excel. If this is not showing up in your version of Excel, we have to first make it show up, by right-clicking on any of the other options at the top of Excel (`Home`, `Insert`, or any other option), and then choosing "Customize the Ribbon". You will get a window, the right side of which contains "Main Tabs" with checkboxes in front of them. Click the checkbox in front of "Developer" and click `Okay`. `Developer` should now be one of the tabs.

Click `Developer`, and `Record Macro`. It is good practice to give all your macros descriptive names, so type something like `Weekdays` and click `OK`. Now you're back in the worksheet. Type `M`, move one cell down by pressing `Enter` (or ↓), type `T`, move one cell down by pressing `Enter` (or ↓), etc., until your first column consists of the five weekdays. Then click `Developer`, and `Stop recording`.

Your macro is saved and ready to be used. Click `Developer` and `Macro` and you will see the name of the macro that you just recorded. Select this macro and click `Run`. Move your cursor to some place where you want the days of the week to appear and run the macro again.

You are probably disappointed that this did not work properly. Let's have a look at what is wrong with this macro. Every macro is translated to a snippet of VBA. You can view the code by clicking `Developer`, then `Visual Basic`. Now double click `Module1`. You should see a snippet of code with the name of your macro.

In this recitation you will write some VBA code yourself, not just by recording macros.

To record the macro that we actually wanted, Click `Developer`, click `Use Relative References` (it will now be highlighted) and then `Record Macro`. Give your new macro a descriptive name and click `OK`. Now you're back in the worksheet. Record the macro as before: Type `M`, move one cell down by pressing `Enter` (or ↓), type `T`, move one cell down by pressing `Enter` (or ↓), etc., until your first column consists of the five weekdays. Then click `Developer`, and `Stop recording`.

When you use your new macro, you will see that it actually has the functionality that you wanted. Have a look at the code of your new macro by again clicking clicking `Developer`, then `Visual Basic` and double clicking `Module1`. You should see both the code of the old and the new macro, and you can try to see if you can understand this code.

2 Writing some more interesting code

This was just a quick example to show you where you can find the VBA code in Excel. You are not expected to be able to write code like this from scratch at the end of this recitation: this recitation is meant to just give you a taste of what you can do in VBA.

We will now explore how VBA code is structured (in particular how to For-loops, While-loops and If-Then statements work), so that you can write VBA code that actually does something interesting.

Open the file `Recitation4.xlsm`. You will probably receive some security warning. If so, click `Enable Editing` and/or something like `Enable Content`. There are a few worksheets with incomplete code that you will finish.

2.1 Factorial Example

Go to the worksheet "Factorial Example". The goal is to make this worksheet do what you expect it to do: when someone clicks `Compute Factorial` the factorial of the number in the green box should appear in the yellow box.

Left-click on the button `Compute Factorial`. Choose "Assign Macro". You will see a list of three macros in this Excel file, one being called "Compute_Factorial". This is the one you will edit to make the worksheet work. Choose that macro and click `Edit`.

You are in the Visual Basic Editor (VBE) again, right at the code snippet that is supposed to calculate the factorial. Most of the code is already there.

Here is something important about the structure of VBA code: every subroutine that is defined starts off with `Sub` and then the name of the subroutine and two parentheses. Next the variables of the subroutine are declared, with the types that they are.

1. What variables are declared in the subroutine "Compute_Factorial" and what are their types?

   After declaring the variables, we get the execution statements. First the variable N receives the value that is in the green box (cell `D4` in the worksheet), and there is a check whether it is a nonnegative value. The we want to calculate $N!$. We will do this using a loop, storing the result in the variable `N_fact`.

2. Use Excel help to find the correct syntax of a for-loop in VBA. Then finish the code snippet so that

it does what we want it to do. Write down the code you added here:

In fact, Excel has a built-in function `Fact` to calculate factorials. You can use built-in functions in VBA by prefacing the function name by `Application.WorksheetFunction`.

3. Modify your code to use the built-in function.

2.2 Doubling Example

The previous example was somewhat lame, because we did not need VBA to get the functionality that we wanted. (The main point was getting used to writing VBA code in Excel.)

Now go to the "Doubling Example" worksheet. Here we want to calculate how many years it takes to double our money for a given yearly growth rate that is $r\%$ (the input in the green cell).

The idea is that if we start with, say, 1 dollar, then at the beginning of the next year we have $1 + r/100$ dollars — our capital has grown with a rate of $r\%$. In general, if we start with d dollars, at the beginning of the next year we have $(1 + r/100)d$ dollars.

We want to write a `While`-loop in VBA that steps through the years, and exits the loop the first time that our capital has doubled, outputting both the years to double, and the total capital we have at that moment, if we started with $\$1$. In the code, the variable `counter` counts the number of years, the variable `c_growth` keeps track of the total capital (cumulative growth).

4. Use Excel help to find the correct syntax of a while-loop in VBA. Then finish the code snippet so

that it does what we want it to do. Write down the code you added here:

2.3 If/Then Statements

Now look at the worksheet "Incrementing Example". You can choose which of the three cells is incremented, by selecting A, B or C and then pressing the button `Increment`.

Have a look at the code to make sure you understand what it does, and then modify it so that the user can increment the cells any amount they want: create a new input cell that specifies the increment, and implement the logic.

2.4 Frequently Used Range Methods

You have seen that you can use `Range(CellName).Value` to get and set the value of a cell (or multiple cells).

Here are some methods (commands) that you can use after `Range(CellName).`:

- `Clear`: deletes everything from the range
- `ClearContents` (`ClearFormats`): deletes the contents (formats) of the range
- `Copy [Destination:= drange]`: copies the range [to the specified destination range]
- `PasteSpecial [Paste:= xlconstant]`: pastes the contents of the clipboard into the range according to the specified criteria
- `Select`: selects the range
- `Sort [Key1:= column, Order1:= xlconstant]`: sorts the range according to the specified criteria

Here are some useful properties that you can use after `Range(CellName).` (these return values):

- `Address`: holds the range address as a string (e.g., "A1:C5")
- `Cells`: used to reference a particular cell in a range (e.g., `Range("A1:C5").Cells(1,2)` references cell B1)
- `Column` (`Row`): holds the number of the first range column (row)
- `Columns` (`Rows`): used to reference a particular range column (row) (e.g., `Range("A1:C5").Columns(1)` references cells A1:A5)

- `Font`: holds properties of the range font such as size, bold, italic (e.g., `Range("A1:C5").Font.Bold = True`)
- `Formula`: holds the range formula as a string (e.g., "`=SUM("A1:C5")`")
- `Interior`: holds the properties of the range interior such as color (e.g., `Range("A1:C5").Interior.Co = 5`)
- `Name`: holds the range name as a string
- `Offset`: used to reference a cell relative to a range
- `Value` (`Value2`): holds the value of the range (default property)

5. Create a worksheet and implement some interesting functionality using some of the properties and methods above, or anything that you find using Excel help.

3 Conclusion

The point of this recitation was to give you the tiniest idea of what you can do in Excel, and, more importantly, take away your apprehension of opening VBE and looking at VBA code. There is all kinds of code that you can download and look at to see what can be done in Excel. Be a little bit careful though: VBA is so powerful that it can be used to create viruses — make sure you only download files from site you trust.

Note that there is a course that teaches how to use Excel for a wide variety of problems in Operations Research: ORIE 4820 Spreadsheet-Based Modeling and Data Analysis. In fact, most of the examples in this recitation where borrowed from the introduction to VBA from this course.