PROCEDURE AND FUNCTION DECLARATION

Exercise 6

Macroinstructions (macroprocedures or simply procedures) start with a keyword **`Sub`** and end with keywords **`End Sub`**. After **`Sub`** you should place the macro's name and a pair of parenthesis. Optionally, input parameters can be declared within the parenthesis. The macro body in contained between **`Sub`** and **`End Sub`**. It contains instructions executed by Visual Basic every time the macro is called.
The VBA expressions usually represent dot-expressions, where dot " . " separates an object from its field or function performed on it, e.g.:

```
Rows("1:1").Select 'select rows from 1 to 1 (in fact it is a single row)
Selection.Insert Shift:=xlDown 'insert a new row on the selected range and
                               'shift the other rows down
Range("B1").Select   'Select cell B1 to add text legend
Selection.FormulaRlC1 = "formulas" 'enter into an active cell "text" value
Range("A1").Select   'Select cell A1 to fill with corresponding color legend
With Selection.Interior 'perform the following operations on the whole
                        'selection's interior
    .ColorIndex = 3 'set color to red
    .Pattern = xlSolid 'set pattern to solid
    .PatternColorIndex = xlAutomatic 'set pattern color to automatic
End With 'end the loop
```

Note that the code above is duplicated 2 times to create legend for constant-numbers (colorindex=4, so green) and constant-text (colorindex=5, so blue). **You can easily understand here that there is a redundancy in code writing and this could be easily replaced by a new macroprocedure**. This is the aim of the following exercise instructions.

Tasks to be done:

- Declare a new procedure called CreateLegend (so write **`sub CreateLegend()`** and the **`end sub`** will appear automatically)
- Cut the code related to legend creation and paste it in the body part of the new procedure.
- Now, in order to be able to execute it, you need to call the procedure in the main procedure, i.e. in FormatTable (or the name you gave to the procedure you recorded). So, where appropriate write **`call CreateLegend.`** Execute the code (press **Run Macro** button from **Standard** tab ).

  WARNING: normally, you should execute first **ClearFormat** to test your procedure. However, be careful with it, because if already executed, you may delete data from the table. But, as programmers, you could later on find a fancy way to make this not possible.

As we said previously, the code is still redundant, because we use 3 times the same instruction block to create the 3 rows of legend in the Excel document. So to simplify this, we will use the possibility to pass variable inputs to the program to simplify it. Indeed, if we analyze the code, we see that **colorindex** value and the string parsed to the **FormulaR1C1** attribute (e.g. "Formulas") correspond to variables. So proceed as follows:

- Keep only one block of code
- add 2 variables between the brackets of the procedure header: one called iColorCell (declare

the datatype as integer; **`colorcell As Integer`**) and the second called sLegend (declared as string).

- Now, in the procedure body, replace where needed the piece of codes by the variables (so where the cell color and the legend corresponding values have been specified).
- In FormatTable, add to the Call statement values of the variables to be parsed, e.g.:
  ```
  Call CreateLegend(3,"Formulas") 'create legend for the Formula resulting in numbers
  ```
- Click anywhere in FormatTable and run. Do you get same results?

This exercise has introduced another VBA aspect, i.e. variable declaration. You have to know that, alike C or Java, variable declaration is not obligatory (depend of case actually). If not declared, they will be automatically set as **Variant**, the VBA mastertype that takes 16 bytes for numbers and 16 bytes per character for strings. So very space consuming. So even optional, it is still a good practice to declare variables. We will see this in next exercise and in more details in lab4.

Exercise 7

This exercise and following ones show you how to create a new spreadsheet and name it, if the name hasn't been taken by another one. It also introduces a bit in more details object hierarchy, function declaration and variable declaration.

First, see below the code of the function **SheetCheck** that checks if a given worksheet exists. Copy its code to the VBA editor:

```
Function  SheetCheck(WB As Workbook,WorkSheetName As String ) As Boolean
Dim  WorkSht As Worksheet
'returns if a worksheet name is an empty string
If  Len(WorkSheetName) = 0 Then
   SheetCheck = True
Exit Function
End If
'checking if a worksheet exists
For Each  WorkSht In  WB.Sheets
   If  WorkSht.Name = WorkSheetName Then
       SheetCheck = False
       Exit Function
   End If
Next  WorkSht
SheetCheck = True
End Function
```

Try to understand the function structure. More importantly, answer the following questions:

- How many variables are parsed to the function?
- What is different between the function structure and the procedure structure?
- Which variable is returned by the function? What is so specific with VBA function declaration?

Exercise 8

Like procedures, functions are called by other functions/procedures. In our case, it will be called in a procedure that is responsible to add a new worksheet to the current Excel document. So copy the

code below and again paste it to your VBA editor.

```
Sub  SimpleSheetAdd()
Dim Workbk as Workbook
Dim  NameNotExist As Boolean
Dim  BookName As String
 BookName = "SomeDocument.xlsm"
'check if a worksheet "new worksheet" exists in ".xlsm"
workbook. Replace SomeDocument by a more clever name (e.g. VBA1_yourname.xlsm)
Set  Workbk = Workbooks(BookName) 'create object
NameNotExist = SheetCheck(Workbk,"new worksheet")
'if a worksheet with name "new worksheet" does not exist, VBA creates it
If  NameNotExist Then
   Workbk.Sheets.Add
   ActiveSheet.Name = "new worksheet"
End If
Set Workbk = Nothing
End Sub
```

Few important remarks:
The previous examples show you the use of object hierarchy in Excel (but we can extend this to any MS Office software). For instance, above you can see that **Workbook** and **Worksheet** are object types, but you can also see that **Worksheet** is a sub-member of Workbook . At the top of this hierarchy is the Application object (here Excel). The "." symbol makes the link between them, e . g . `Application.Workbooks("Book1.xlsx").Worksheets("sheet1").Range("A1").Value`, which can be read as "the value of the cell "A1" in the worksheet "sheet1" of the Excel document "book1.xlsx". You can easily think of the hierarchy in MS Word also.

Again, you are not always forced to declare object variables in VBA (like standard variables) but you are highly advised to do so, first for good programming habit, and also for practical reasons: you may end with handling in one program several excel documents that will have to communicate data between each other. You can easily understand that declaring object variables may significantly reduce typing long hierarchical expressions as the one above. To do so, you will have to:

- declare object variable, usually at the beginning of the procedure/function (see examples above)
- initialize the object using the **Set** statement (see also example above).
- It is also good practice to remove from memory an object when not used using the `Set objectname= Nothing` statement (see again above).

Last but not least, the code also shows example of standard variable declarations, i.e. of type **boolean** or **string**. Even not compulsory, it is advised again to declare. Why? Because VBA is a interpreted language, it does not check by default variable declaration. This has some advantages of flexibility. However, this is prone to error (especially spelling error). If a variable contains a spelling error, it is treated as a new variable and therefore will cause syntax error. If you want to force VBA to declare all variables, you will have to write at the top of your module (before any procedure/function declaration, the `Option Explicit` statement.

Run the code above. What Happen? Any problem?

Exercise 9

Add to the code from the previous exercise some functionality so that it asks a user to enter the new worksheet name. Use:

```
NewName = InputBox("Enter a worksheet name:", "Add new worksheet", "whatever")
```

where **NewName** should be declared as:

```
Dim NewName As String
```

The prompt will appear:

Exercise 10

Create a macro prompting for a name of a worksheet to delete and then deleting it (if exists). If a worksheet does not exist, an appropriate dialog should be displayed. Use the code:

```
If Not NameNotExist Then
Workbooks(BookName).Sheets(NameToDelete).Activate
ActiveWindow.SelectedSheets.Delete
Else
MsgBox "Worksheet" & NameToDelete & " not found...", vbOKOnly, "Warning"
End If
```

The two last exercises have introduced 2 popular VBA built-in functions: M**sgbox** and I**nputbox. Msgbox** displays a message to the screen while I**nputbox** is an interrogation window waiting for an input to be parsed for a (declared or not declared) variable. Also, in the M**sgbox** first field, see the use of the ampersand "**&**" as string concatenation operator. These are popular Visual Basic functions and language symbols. A broader view of the language structure will be approached in the next lab.