Excel VBA: Working with Excel Function & Data

Excel VBA programming - Part 3

Harun Kaplan

bookboon
The eBook company

HARUN KAPLAN

EXCEL VBA: WORKING WITH EXCEL FUNCTION & DATA

EXCEL VBA PROGRAMMING - PART 3

Excel VBA: Working with Excel Function & Data: Excel VBA programming - Part 3
1st edition

ISBN 978-87-403-2412-9

CONTENTS

INTRODUCTION

So far we have covered:

In the first book “Excel-VBA Introduction”: General Information, Editor Environment and Language Concept of VBA Programming;

In the second book “Excel-VBA Working with Excel Elements”: Working with Workbooks; Worksheets; Range / Cells; functions; Comments, etc.

The third book “Excel VBA Working with Excel Functions & Data” is mainly about

- Excel functions in VBA (reference, CountIf, Sum, Min, Max)
- Programming your own functions
- Converting and manipulating data
- Creating your own menu
- Graphic creation
- Windows Explorer features with VBA

The intended target group for this book includes beginners as well as advanced users.

Harun Kaplan

For my Family:
Tülay
Yasin, Sueda, Melik

1 STATEMENT AND VBA-FUNCTION

A program is controlled by the instructions or functions. VBA offers us a large number of built-in features:

- Deliver values,
- To transform values,
- To pass on to another function,
- To manipulate or evaluate strings
- Etc.

We will later learn to write our own functions. Here we would like to familiarize ourselves with the Excel functions in VBA, which we know in Excel as "Formulas | Insert Function".

There are many functions included in Excel. For example:

Excel	VBA (WorksheetFunction)
CountIf	.CoundIf(Range(„A1:A65535"),1)
VLookUp	.VLookUp(123,A1:C100,3,False)
Count	.Count(Range(„A1:A65535")) oder SubTotal(2,A3:A32)
SubTotal	.SubTotal(103,A3:A65535)
SumIf	.SumIf(Range(„A:A"), „>25")
SubTotal	.SubTotal(3,Range(„A:A"))
Max	.Max(Sheets(„Table1").Range(Cells(2,3),Cells(65536,3))
Min	.Min(Sheets(„Table1").Range(Cells(2,3),Cells(65536,3))
Sum	.Sum(Range(Cells(1,1),Cells(5,1))
CountBlank	.CountBlank(Range(„A:A"))

Table 1: Important functions - WorksheetsFunction -

That is where we enter a function in VBA editor:

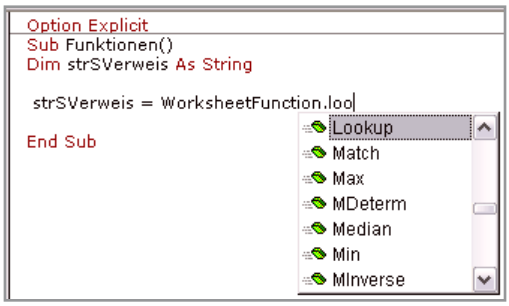


Figure 1: Insert WorksheetsFunction in VBA

1.1 SEARCH WITH VLOOKUP-FUNCTION

VlookUp is a powerful tool. It can search for a value in a foreign table. It finds a search value in a range and returns the value of an associated column. The value being sought out must always be in the first column of the specified range. It is important to know that the column index should always be counted from the position of the search column.

VLookUp looks for a value in the left-most column of a table, and then returns a value in the same row from the column you specify.

It is typed in VBA code as **WorksheetsFunction.Vlookup** ().
It will use in the following cases:

- Target and source tables in the same file
- Target and source tables in different open files
- Opened target file and closed source file

We first write out VLookup in words.

1. Definition of the result, whether it is written to a cell or assigned to a variable: output_in_cell *"Result"*.
2. Then follows the **WorksheetFunction.VLookup** statement
3. The figures that should be compared checking_in_column "*Control_Column"*
4. How big the comparison matrix control mask is "*Control_Mask"*
5. If found, which cell of the number column of the comparison matrix is output "*Nr_Column*"
6. Finally, the truth value FALSE for the exact match to take place.

*Result=**WorksheetFunction.VLookup**(Control_Column,Control_Column, Nr_Column)*

Let's take a look at the examples:

1.1.1 TARGET OR SOURCE SHEET IN SAME FILE

For this example, we have a file with multiple tables. The content of cell "A20" from "Sheet1" shall be compared with the table "Cars" in the range A1: B20 and the result entered in cell "B20" of "Cars".

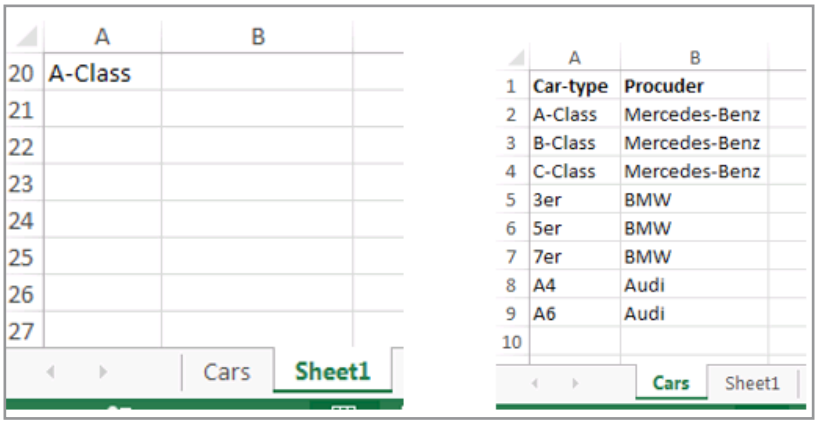


Figure 2: Target and source sheet in same file

```
Sub Example_VLookUp_1_intern_source_file_open()
Dim wksTarget As Worksheet
Dim strSearch As String
strSearch = Worksheets("Sheet1").Range("A20").Value
  Set wksTarget = Worksheets("Cars")
    Range("B20").Value = WorksheetFunction.VLookup(strSearch, _
    wksTarget.Range("A1:B20"), 2, False)
End Sub
```

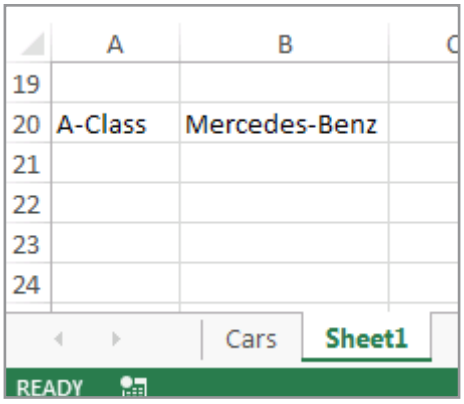


Figure 3: Result of VLookUp_1

1.1.2 TARGET OR SOURCE SHEET IN DIFFERENTLY FILE

Here we have the same example as before. However, our comparison matrix is in a different file and it is open so both files are open.

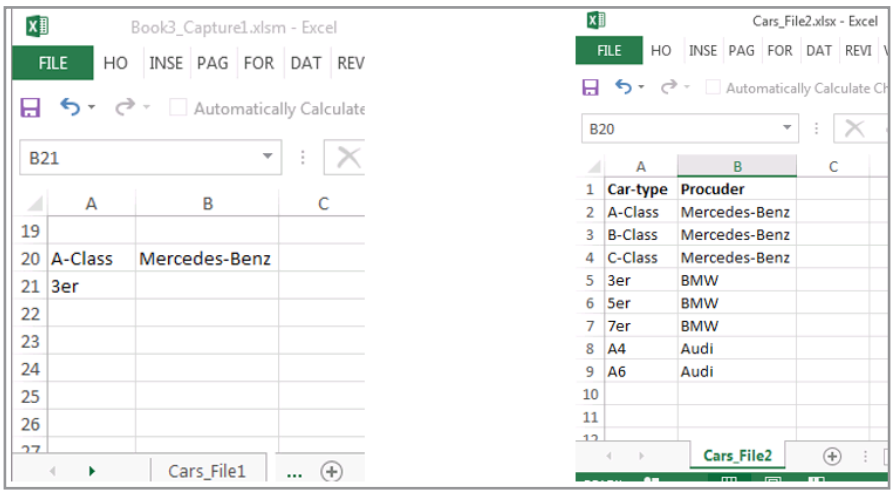


Figure 4: Target file and Source file "Autos.xlsx"

```
Sub Example_VLookUp_2_extern_source_file_open()
Dim wksTarget As Worksheet
Dim strSearch As String
strSearch = Worksheets("Sheet1").Range("A21").Value
  Set wksTarget = Workbooks("Cars_File2.xlsx").Worksheets("Cars_File2")
    Range("B21").Value = WorksheetFunction.VLookup(strSearch, _
    wksTarget.Range("A1:B20"), 2, False)
End Sub
```

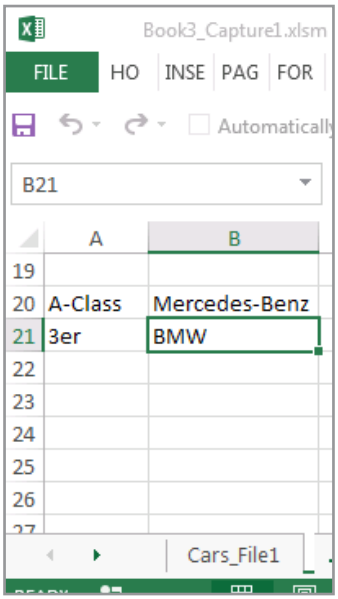


Figure 5: Result of VLookUp_2

1.1.3 TARGET SHEET IN OPENED FILE AND SOURCE SHEET IN CLOSED FILE

Again, we have the same example as before. This time the comparison matrix is in another file and it is closed. I think this variant is also used frequently. We will use a little trick here.

To do this, we insert an "Excel reference function" in the cell and then enter the content as a value. In our example, it would also be necessary to check if the file exists. We'll see that later. Here we assume that the necessary file exists.

```
Sub Example_VLookUp_3_extern_source_file_closed()
  Range("B22").FormulaLocal = "=VLookUp(A22;'C:\Kaplan\Kitap\_corrected\
                                [Cars_File2.xlsx] Cars_File2'!$A:$B;2;0) "
  Range("B22").Value = Range("B22").Value
End Sub
```

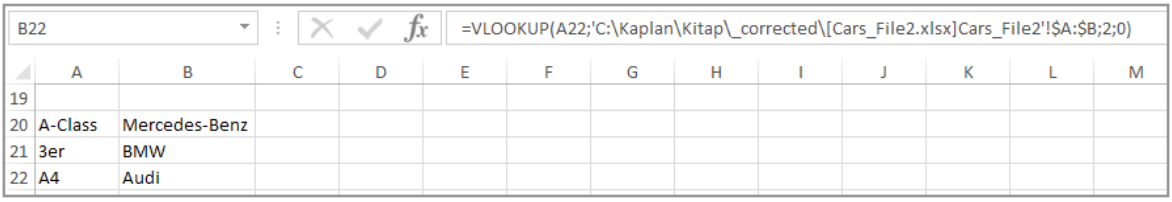


Figure 6: Result VBA_VLookUp_3 => "added Formula"

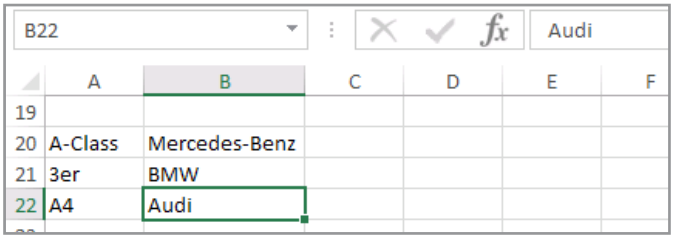


Figure 7: Formula converted into value.

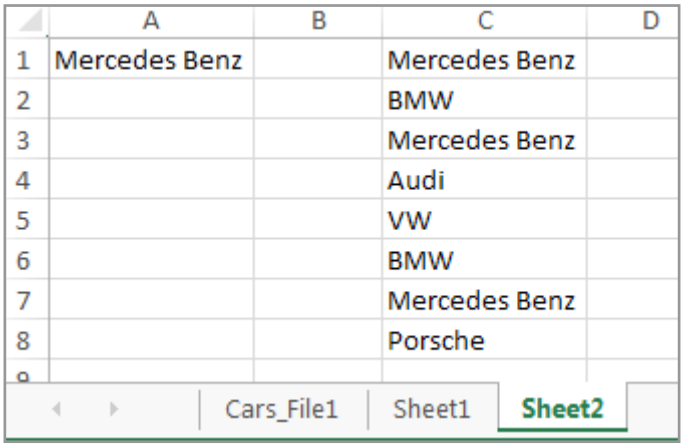
A VLOOKUP-Function with variables looks like this:

```
i = 0
 intRow = ActiveCell.Row
  For y = 3 To 14
    ActiveCell.Offset(0, i).FormulaLocal = "=SVERWEIS(A" & intRow & ";'Sheet1'!$C:$P;" & y & ";0)"
      If ActiveCell.Offset(0, i).Value < 1 Or ActiveCell.Offset(0, i).Value = "" Then
        ActiveCell.Offset(0, i).Value = ""
      Else
       ActiveCell.Offset(0, i).Value = ActiveCell.Offset(0, i).Value
     End If
      i = i + 1
Next y
```

1.2 COUNTIF-FUNKTION

That is also an important function. The function "**.CountIf**" counts the non-empty cells of an area whose contents match the search criteria.

In our example the content of the cell "A1" in the column "C: C" is searched for and the number of it is output.



	A	B	C	D
1	Mercedes Benz		Mercedes Benz	
2			BMW	
3			Mercedes Benz	
4			Audi	
5			VW	
6			BMW	
7			Mercedes Benz	
8			Porsche	

Cars_File1 | Sheet1 | Sheet2

Figure 8: Sheet for .CountIf

```
Sub CountIf_Function()
Dim rngAddress As Range
Dim intCount As Integer
Dim strSearchobject As String
  strSearchobject = Cells(1, 1).Value
   Set rngAddress = Worksheets("Sheet2").Range("C:C")
     intCount = WorksheetFunction.CountIf(rngAddress, strSearchobject)
       MsgBox "the search term - " & strSearchobject & " - occurs " & intCount & "x."
End Sub
```

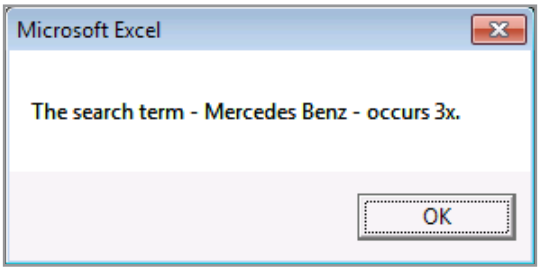


Figure 9: Result of .CountIf

1.3 SUM-, MIN- UND MAX- FUNCTIONS

We can also express the summation and min and max values in VBA. Suppose we have a lookup table as shown below.

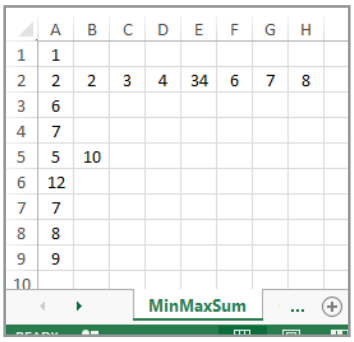


Figure 10: Sheet for Sum / Min / Max

We want here

- The sum of A2 to A4 + B5
- The sum and min-value of the values of row 2:2
- The sum and max-value of the values of column A:A

```
Sub Mathe()
 Dim intSum_of_cells, intSum_of_row, intSum_of_column As Integer
 Dim intMin_Value, intMax_Value As Integer
 intSum_of_cells = WorksheetFunction.Sum(Range("A2:A4", "B5"))
 intSum_of_row = WorksheetFunction.Sum(Rows(2))
 intSum_of_column = WorksheetFunction.Sum(Columns(1))
 intMin_Value = WorksheetFunction.Min(Rows(2))
 intMax_Value = WorksheetFunction.Max(Columns(1))

 MsgBox ("The sum of all cells: " & intSum_of_cells & Chr(10) & _
          "The sum of row: " & intSum_of_row & Chr(10) & _
          "The sum of column: " & intSum_of_column & vbLf & _
          "Min Value: " & intMin_Value & vbLf & _
          "Max Value: " & intMax_Value & vbLf)
End Sub
```

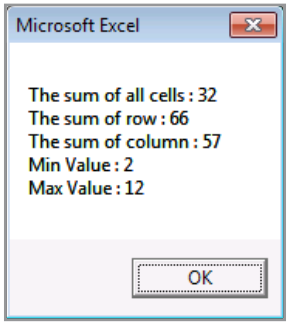


Figure 11: Result Math()

1.4 OWN FUNCTION

All functions work according to the "Input | Processing | Output" principle. To be able to get a result something, such as a value, must be entered. This value is edited and output. In other words, input - processing - output. The sheets and VBA functions also expect transfer values with which a calculation can be performed. The result is shown in the desired cell or as a message box or printed.

1.4.1 CREATE YOUR OWN FUNCTION

In the first example we will write a simple procedure. Then we will transform it into a function. Later we will insert this function in the sheet or in another procedure as a function.

In this example, two values will be read. They are added up together. The result is then entered in a cell. Here three variables in type "Double" were used. They are declared one after the other at the beginning of the procedure with the Dim statement.

The values to be added up are read from different cells, here "A2" and "B2", and the result is entered in cell "C2".

```
Sub Example_1_Function()
Dim dblValue_1, dblValue_2, dblResult As Double
  dblValue_1 = Cells(2, 1).Value
  dblValue_2 = Cells(2, 2).Value
  dblResult = dblValue_1 + dblValue_2
  Cells(2, 3).Value = dblResult
End Sub
```

We are already familiar with this. Now to the function:

A function procedure is written between **Function ... End Function**. We can rewrite the example above and make it a function. Then we use it in an Excel sheet.

- Accessible under "**Formulas | Insert Function | User Defined**".

The name of the function is in our example "dblResult". Additional variables are declared individually between the parentheses. This tells us that the result from the variables is determined in brackets.

```
Function dblResult(dblValue_1 As Double, dblValue_2 As Double) As Double
        dblResult = dblValue_1 + dblValue_2
End Function
```

1.4.2 USE YOUR OWN FUNCTION IN EXCEL SHEET

Now the value "34" is in cell "A3" and the value "45" is in cell "B3". The result should be calculated in cell "C3" by a function. We click on cell "C3" and insert our function "dblResult" in the menu "**Formulas | Insert Function**".

Since the result is calculated from two values, two input fields for intValue_1 and intValue_2 appear.

This is how it looks when this function is inserted into the cell of a table:

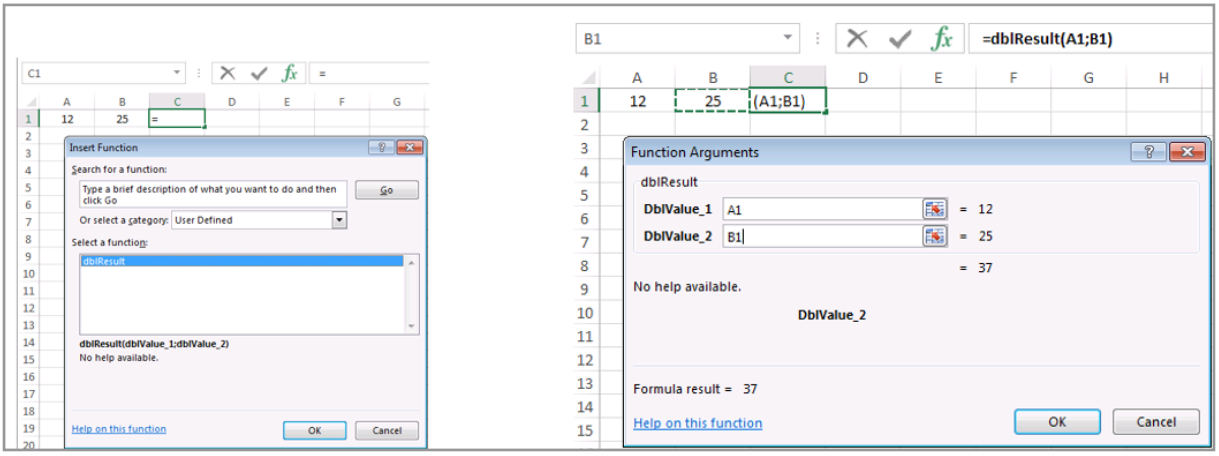


Figure 12: Own function in Excel

1.4.3 USE YOUR OWN FUNCTION IN SUB-PROCEDURE

We can also use it in a procedure. Then she would look like this:

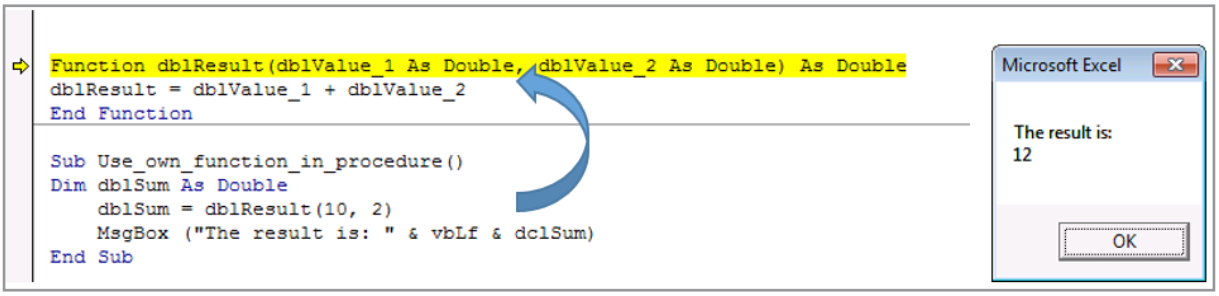


Figure 13: Own function in Sub-Procedure

If we let the macro "Own_Function_Application_in_Procedure ()" step by step, i.e. with the function key "F8" step by step, it jumps from the third line in the "Function intResult" and then runs through there.

1.5 UNLIMITED NUMBER OF ARGUMENTS

In such cases we can work with the "**ParamArray**" function. It allows us to specify an unlimited number of arguments. It should be noted that Variant is used as the data type.

```
Function dblResult2(ParamArray varPArray() As Variant) As Double
  Dim varValues As Variant
    For Each varResult In varPArray()
      dblResult2 = dblResult2 + varValues
    Next varValues
End Function
```

Well, what did we do here?

1. In the first line (header) enter the function "**ParamArray**" with the variant declaration.
2. In the second line, the result "varValues" is declared as a variant.
3. From the third to the fifth line, the "For Each loop" goes through all the "varValues" and adds them together.
4. At the end "End Function".

It will then look like this if it is inserted in the table:

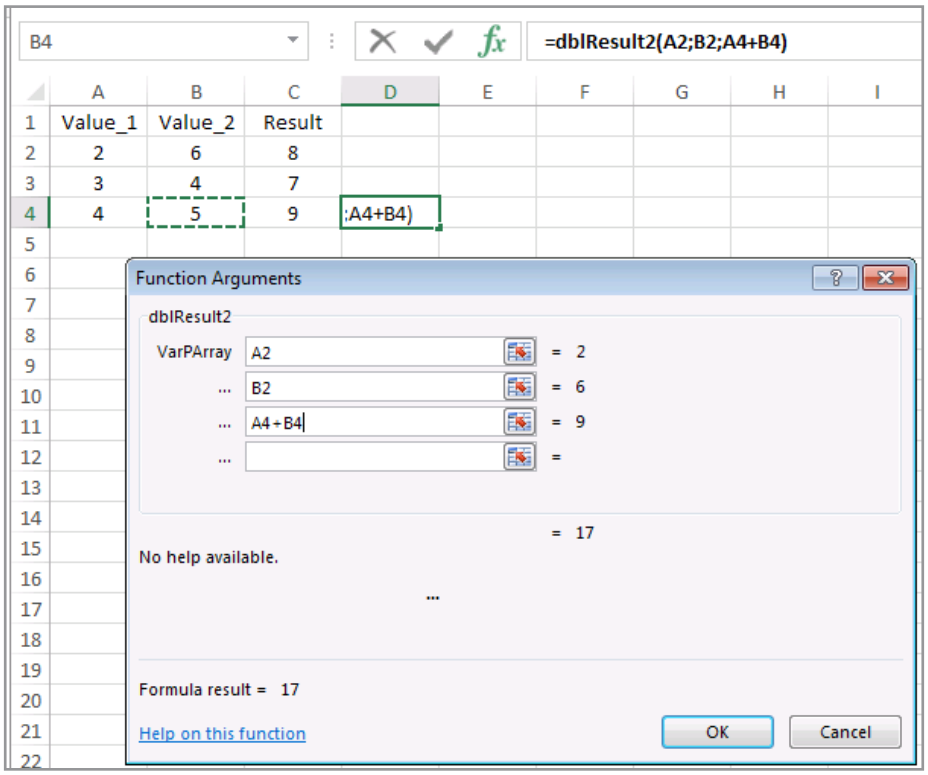


Figure 14: Function with more parameters

So far we have seen individual cells in unbound areas. Now we will create a function with areas.

1.5.1 RANGE SPECIFICATION IN A FUNCTION

In our next example, we will create a function to read out, for example, the maximum value of a range. This area is declared as "Range".

```
Function dblMaxValue(rngBereich As Range) As Double
  dblMaxWert = Application.WorksheetFunction.Max(rngBereich)
End Function
```

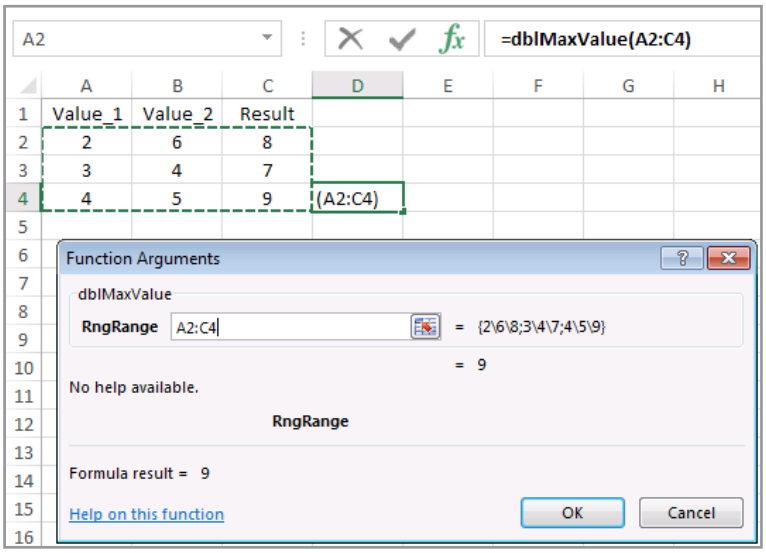


Figure 15: Function with range specification

So far, our examples have been presented with at least one argument. Now an example without arguments.

1.5.2 FUNCTIONS WITHOUT ARGUMENTS

In our example, we will set up our page for printing with "file with path" in the footer. Thus, a value is not absolutely necessary.

We will print it using the current file name with text in front of it. The result is entered in the next step "Macro7" in the header in the right section.

```
Function strFilename() As String
  strFilename = "Filename is " & ActiveWorkbook.Name
End Function

Sub Makro7()
   With ActiveSheet.PageSetup
      .LeftHeader = strFilename
      .LeftFooter = "&A"
   End With
End Sub
```

1.5.3 ASSIGNING A FUNCTION TO A CATEGORY

By default, our features are listed in the Custom category. We can assign them to the category index of a different category using the "**MacroOptions**" method.

The main categories look like this:

Index	Categories
1	Financial Mathematics
2	Date and Time
3	Mathematics & Trigonometry
4	Statistic
5	Matrix
6	Database
7	Text
8	Logic
9	Information
14	User definition

Table 2: Category-Index

In the procedure below "dblResult" is assigned to the category Mathematics & Trigonometry.

```
Sub Category_assign()
Application.MacroOptions _
        Macro:="dblResult", _
        Category:=3
End Sub
```

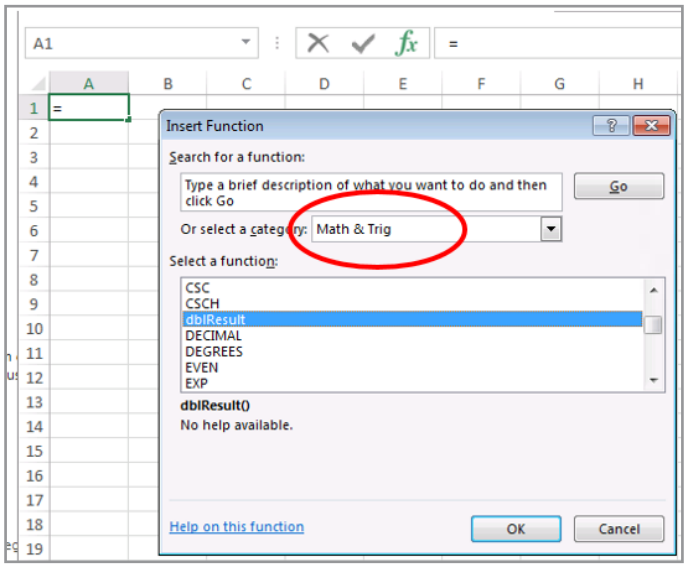


Figure 16: Function in Catagory "Math.&Trigonom."

It would also be nice if we could find our functions under our defined category. That is also possible. We can assign them to the "My Features" category so they are all in one location.

```
Sub Category_assign()
Application.MacroOptions _
        Macro:="dblResult", _
        Category:= "My Functions"
End Sub
```

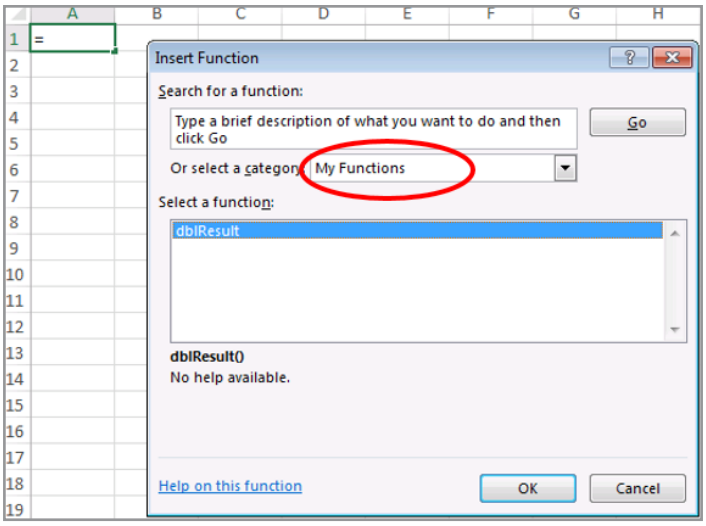


Figure 17: Category " My Functions "

1.5.4 ASSIGN A DESCRIPTION TO A FUNCTION

Almost all standard functions of the Excel have a help text. It describes what can be achieved with each function. For self-created functions, the default text is "No help available."

We can create a help text for our function with the "**Description**" property of the "**MacroOption**" method.

```
Sub Category_assign_dblResult()
Application.MacroOptions _
  Macro:="dblResult", _
  Description:="This sums up two values." & vbLf _
               & "First select the first value, then second value.", _
  Category:= "My Functions"
End Sub
```

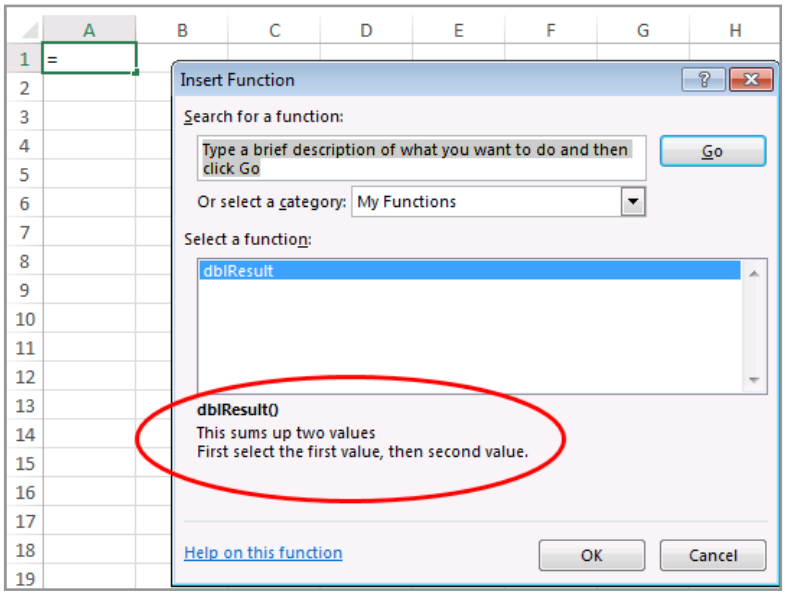


Figure 18: HelpText to a function

2 CONVERTING AND MANIPULATING DATA

When a procedure is created, "**Option Explicit**" always appears. In this case, we must define all variables in the procedures with accurate matching data types. If something is missing an error message will appear. If we delete "Option Explicit" or deactivate it with an apostrophe, undeclared variables are automatically declared with the data type "**Variant**".

For programs to function properly it is important to know what type of data the variables used should have. If these declarations are incorrect or undefined it may cause runtime errors.

If you work on a large project with different programming then such definitions are necessary.

2.1 DATA TYPES

All the data or information that we enter in Excel belongs to certain types of data. They can be numbers or texts. This section is about dealing with such elements. In some cases these types may determine date or convert date from one type to another type.

2.1.1 DETECTION OF DATA TYPES

If necessary we can also determine the data types of the variables. For this we need the function "**VarType** ()". In doing so, we not only determine the data type of a variable, but also the type of expressions of, for example, a range object.

We write a text in the cell "A1", then we run the lower macro. As a result, we get the value 8, which is a string. If we overwrite the cell "A1" with a numbe our result will be 5, so it becomes a floating-point number with double precision.

We get a list with all return values in the VBA help of the VarType function.

```
Sub Variable_retrieval2()
 Dim intVariable As Integer
   intVariable = VarType(ActiveSheet.Range("A1"))
End Sub
```

2.1.2 CONVERSION OF DATA TYPES

Data types entered in the cell, i.e. number, text, etc. can be re-keyed among each other. That is, we can re-type a number in text or the text in number if a number is typed as text.

Our next two examples show us how

- a number in text, i.e. from integer to string and
- a text in number, from string to integer.

2.1.3 CONVERTING A STRING IN A NUMERICAL VALUE

It is not to be forgotten that lists may be exported from the databases. It is also possible happen that some numbers are transmitted as texts. In such cases, re-typing is indispensable.

This conversion of a numeric value into strings is possible by means of the functions "**Val**".

The first option is:

```
Sub variable_text_in_number_1()
    Range("a1").Value = Val(Range("a1").Value)
End Sub
```

The second way to do this is by multiplying the contents of the cell by 1:

```
Sub variable_text_in_number_2()
   Range("a1").Value = Range("a1").Value * 1
End Sub
```

If we have several number-looking texts then they can all be re-typed using a for loop. In our example we will change from "A1" to "A10". As a control, we can create a molecular formula in cell "A12" to see if they have been correctly re-typed. If the values add up, then we are on the right track. ☺

```
Sub variable_text_in_number_3()
Dim i As Integer
  For i = 1 To 10
    Cells(i, 1).Value = Cells(i, 1).Value * 1
  Next
End Sub
```

2.1.4 CONVERTING A NUMERICAL VALUE IN A STRING

To do this the other way around, you can use the functions "**Str**" and "**CStr**".

```
Sub number_in_text1()
   Range("a1").Value = Str(Range("a1").Value)

    'Or
    'Range("a1").Value = CStr(Range("a1").Value)
End Sub
```

2.2 MANIPULATING DATA

In this chapter we will see some ways to "manipulate" our data. These are also areas that we will often use in practice.

2.2.1 CUTOUTS FROM A STRING

We can use the functions "Right", "Left" and "Mid" to extract excerpts from a text. The result will then be declared as a "string".

Right *(text, length)* releases a part out of a text, starting from the right;

Left *(text, length)* releases a part out of a text, starting from the right. It will be as many characters as indicated in the value.

Mid *(text, start position, [length])* releases a part of a text, starting with the specified position to the right by number of characters specified in "length". If we did not specify LENGTH, all characters to the right of "Start Position" will be displayed.

```
Sub Example_string()
 Dim Text As String
   Text = "Example String"
     Debug.Print Right(Text, 8)
     Debug.Print Left(Text, 4)
     Debug.Print Mid(Text, 6, 4)
     Debug.Print Mid(Text, 5)
End Sub
```

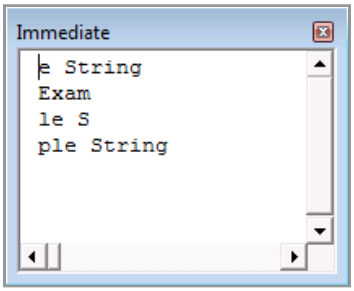


Figure 19: Example of Right; Left; Mid

2.2.2 TEXT LENGTH

With the "**Len**" function we can determine the length of a text. The result is declared as "Long".

The example below shows texts

- from a variable and
- from the cell "A1"

read.

```
Sub Text_length_with_Len()
Dim lgStringlength_Range As Long
Dim strText As String
   'String from Variable
   strText = "C:\HKAPLAN\Bücher\Excel\Arbeitsmappe_Workbooks.xls"
   'String from cell A1
   lgStringlength_Range = Len(Worksheets("Sheet2").Range("A1"))
   Debug.Print Len(strText)
   Debug.Print lgStringlength_Range
End Sub
```

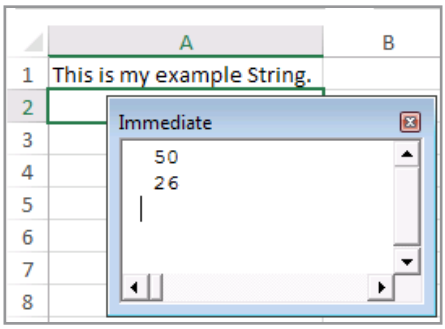


Figure 20: Result Text length with Len

The next example is about a text from a cell. From the entire text length, five characters from the right or left side are cut off and rest is output.

```
Sub Stringlength()
Dim strLength, strText1, strText2 As String
  strText1 = Right(Range("A1"), Len(Range("A1")) - 5)
  strText2 = Left(Range("A1"), Len(Range("A1")) - 5)
    MsgBox strText1 & Chr(10) & strText2
End Sub
```

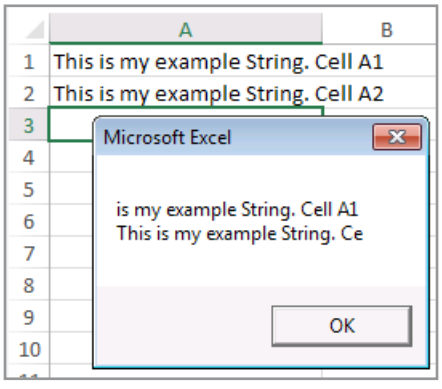


Figure 21: Result "Stringlength"

2.2.3 FIND A SPECIAL CHARACTER IN A STRING

We can find the first and last positions of a given character in a string. The following functions are responsible for this:

- **InStr** function finds the first character from the left and
- **InStrRev** function finds the last character from the right.

strText = "C:\HKAPLAN\Books\Excel\Workfolder_Workbooks.xls"

The character you are looking for is in the third position from the left.

However, if the last position counted from left to right of a particular character in a string is to be determined, we will use the **InStrRev** function.

strText = "C:\HKAPLAN\Books\Excel\Workfolder_Workbooks.xls"

The character you are looking for is numbered 24th from the left.

Both results are declared as long.

```
Sub Example_InStr_InStrRev()
Dim lgInStr, lgInStrRev As Long
Dim strText As String

strText = "C:\HKAPLAN\Bücher\Excel\Arbeitsmappe_Workbooks.xls"

lgInStr = InStr(1, strText, "\")
lgInStrRev = InStrRev(strText, "\")

'String from Cell "A1"
'lgInStr = InStr(1, Range("A1").Value, "\")
'lgInStrRev = InStrRev( Range("A1").Value, "\", , vbTextCompare)

Debug.Print "The first character you are looking for is at the " & lgInStr & ". position."
Debug.Print "The last character you are looking for is at the " & lgInStrRev & ". position."
Debug.Print Right(strText, Len(strText) - InStrRev(strText, "\"))
End Sub
```

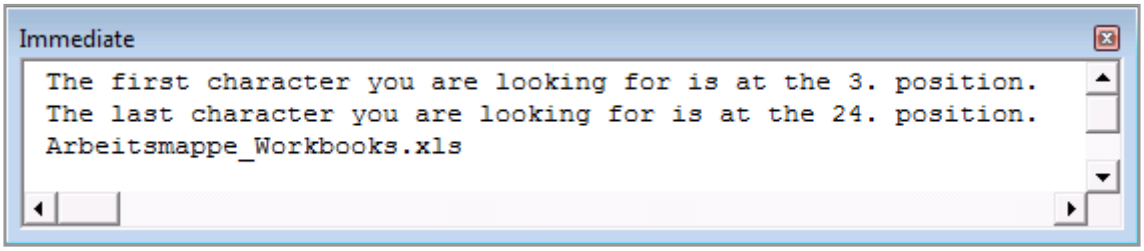


Figure 22: Example InStr, InStrRev

2.2.4 REMOVE SPACES IN A STRING

Suppose we have imported some information from a foreign source. The texts contain unnecessary spaces in front of or behind the text.
With the following functions we can remove unnecessary spaces at different positions in a text.

- **RTrim** removes right-sided, trailing spaces,
- **LTrim** removes left leading blanks
- **Trim** removes right and left or leading and trailing spaces.

The result is declared as a string.

As a sample text, I took my name with two spaces on the right and on the left.

```
Sub Remove_of_unnecessary_Texts()
Dim strText As String
  strText = "  Harun Kaplan "
  Debug.Print RTrim(strText) 'Removes right-sided blanks
  Debug.Print LTrim(strText) 'Removes left-sided blanks
  Debug.Print Trim(strText) 'Remove right-sided and left-sided blanks
End Sub
```

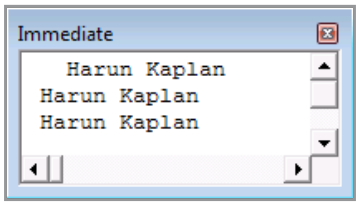


Figure 23: Result "Remove of unnecessary Texts"

2.2.5 CONVERTING STRING IN UPPERCASE OR IN LOWERCASE

It may be that some texts should only be entered in capital letters. The following functions can be used to convert text to uppercase or lowercase:

- **UCase** conversion to capital letters and
- **LCase** conversion to lower case.

The result is declared as a string.

```
Sub Example_UCase_LCase()
 Dim strText As String
  strText = "Harun Kaplan"
   Debug.Print UCase(strText) 'convert to uppercase
   Debug.Print LCase(strText) 'convert to lowercase
End Sub
```

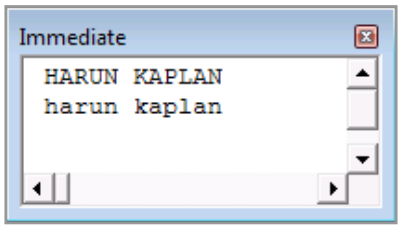


Figure 24: Result Example_UCase_LCase

2.2.6 SPLITTING OF STRINGS

Sooner or later we will get data either from a database, from the internet or from another source with a certain character. We can do this with the function "**Split** *(cell," character ")"* on that character. For example, semicolon "**;**".

Our cursor is in cell "A1" of the sheet (Sheet1). The content of it declared as strText.

strText = "Muster; Mustermann; Musterstrasse; Musterstadt; Musterland" is six entries separated by a semicolon.

StrText should be split at ";" digits and entered in column "A" from blank space. Our VBA code looks like this:

```
Sub spliting()
strText = ActiveCell.Value
With Worksheets("Sheet1")
   varSplit = Split(strText, ";")
   For i = 0 To UBound(varSplit)
      Cells(Rows.Count, "A").End(xlUp).Offset(1, 0).Value = varSplit(i)
   Next i
End With
End Sub
```

And the result for them:

	A	B	C	D	E
1	Muster;Muster	mann;Musterstrasse;Musterstadt;Musterland			
2	Muster				
3	Mustermann				
4	Musterstrasse				
5	Musterstadt				
6	Musterland				
7					

Figure 25: Example for Splitting

2.2.7 CONTROLLING OF SPECIAL CHARACTER

Now let's see how certain special characters can be controlled. This is necessary if the results are to be automatically saved under new names or automatically assigned to directory names because they may not contain special characters.

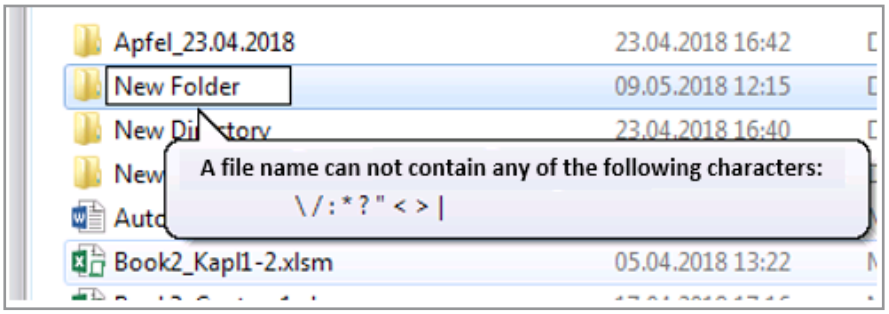


Figure 26: Example special characters

In this case, we can check whether a read or entered character contains a special character.

A list of special characters are: "\", "/", ":", "*", ">", "<", "[", "]", "|".

In our example the text from the cell "A1" should be input and, together with the data, a file name should be generated.

After reading the contents of the cell-A1, all characters of the text are individually checked for special characters. If special characters are found, we will receive a message indicating which character this is.

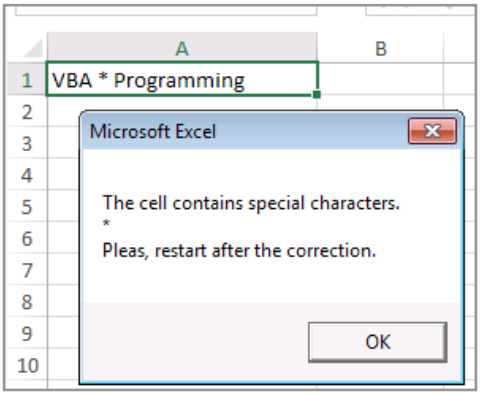


Figure 27: Result Control of special character

Finally, the length of the "filename" is controlled, or "counted". If it contains more than 15 characters, we will receive a message again.

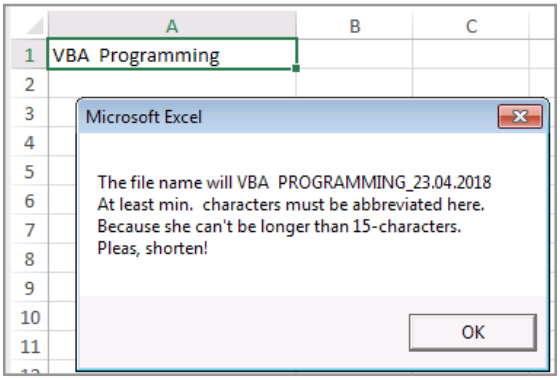


Figure 28: Result after the counting of the text

Well, let's get to our VBA code of the example:

```
Sub Make_Directory()
 Dim strContent, strFilename As String
 Dim intLength_2 As Integer
 Dim lngLength As Long

 strContent = Worksheets("Sheet1").Range("A1").Value
 strFilename = strContent & "_" & Date
  For lngLength = 1 To Len(strContent)
    Select Case Mid(strContent, lngLength, 1)
     Case "\", "/", ":", "*", ">", "<", "[", "]",”|"  'List of special characters
       MsgBox "The cell contains special characters." & Chr(10) _
       & Mid(strContent, lngLength, 1) _
       & Chr(10) & "Pleas, restart after the correction."
          Exit Sub
    End Select
 Next
 If Len(strFilename) > 15 Then
    lngLength_2 = Len(strFilename) - 15
      MsgBox "The file name will " & UCase(strFilename) & vbLf & _
          "At least min. " & intLaenge_2 & " characters must be abbreviated here." & Chr(10) & _
          "Because she can't be longer than 15-characters." & Chr(10) & _
          "Pleas, shorten!"
         Exit Sub
 End If
End Sub
```

2.2.8 INDEXING

Indexing is a way of counting records. For example

First record
2nd Record
3rdRecord
4th record, etc.

In VBA, it is written like this:

Data (i)

This data record must be declared dynamically in the decleration area with ReDim data record (10). Such declarations are given in brackets.

The ReDim statement reserves space to set or resize a dynamic data field.

Examples:

```
ReDim Data_series(5)        ' Reserve 5 items.
For I = 1 To 5              ' Loop 5 ties.
   Data_series (I) = I      ' Initialize data field.
Next I
```

As long as we go through less than 10x here there is no problm. But if we go through more than 10x we will get an error message.

We will now write an example with the above function.

```
Sub Example_make_data_field ()
ReDim Data_series(15)
For i = 1 To 5
   ActiveCell.Offset(i, 0) = i & Data_series(i) & ". Data row"
Next i
End Sub
```

The result for them:

	A
1	Data row
2	1. Data row
3	2. Data row
4	3. Data row
5	4. Data row
6	5. Data row

Figure 29: Result indexing

2.2.9 CONCATENATION OF TEXTS

The concatenation operators are used to join strings. Both alphanumeric and mixed, alphanumeric and numeric expressions are used.

Using the concatenation operator &, variables and constants can be linked as required.

```
Sub Concatenation()
 Dim strName, strAreacode, strNr As String
 Dim intNr_2 As Integer
  strName = "Harun "
  strAreacode = " 0711 "
  strNr = " 555 1234 "
  Nr_2 = 2017

   Debug.Print strName & ", Tel.: " & _
               strAreacode & " - " & Nr & _
               "Year : " & Nr_2
End Sub
```

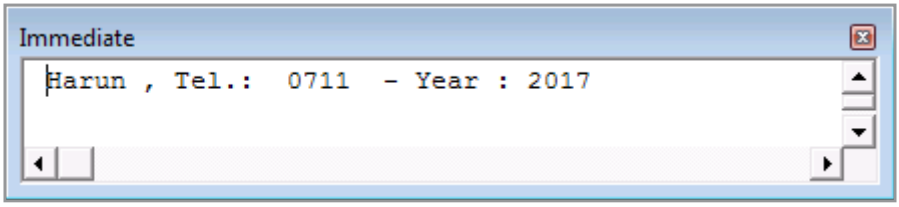


Figure 30: Result concatenation

3 WINDOWS-EXPLORER-FUNCTIONS WITH VBA

With VBA, we can communicate not only with the Office applications, but also with the functions of the Windows Explorer address. For example; create, delete, move, copy, check for, or rename a directory or file.

We are here to learn the functions from Windows Explorer with VBA listed below:

- a File
  - Open,
  - Create
  - Move / Copy / Delete and
  - Rename by “Save as”.
- a directory / subdirectory
  - Create / Delete,
  - Change
  - Check for presence.
- Read system information

We will implement our examples of Explorer functions with File-System-Object-Model (FSO). The FSO model offers an object-based toll for working with folders and files. To use the FSO in VBA, we need to provide some references in the library in VBA.

3.1 MICROSOFT SCRIPTING LIBRARY

In VBA we have three different libraries available. These enable us to access the external files via VBA via the File-System-Object.

In order to be able to use a desired library, it must be activated in the VBA environment of the Excel in the menu item **TOOLS / REFERENCES**, as shown in the figure.

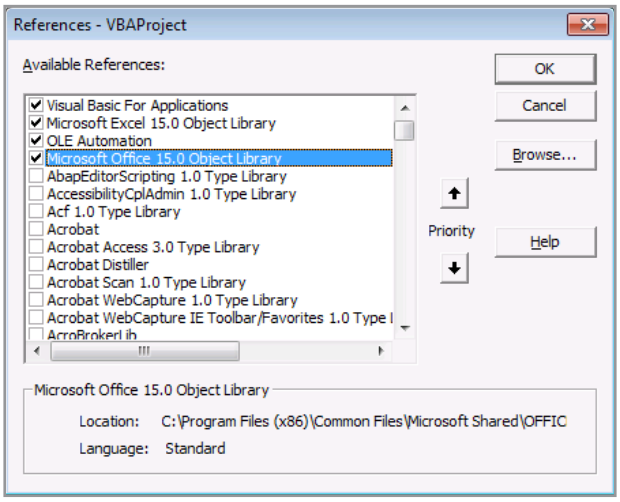


Figure 31: Selection about References

From now on, we will look at useful and practical examples with FSO.

Now we can start by opening or closing a file. ☺.

3.1.1 OPEN OR CLOSE A FILE

We already got to know the opening of a file or a workbook in previous books. In our first example we would like to enter the file directly with the respective path.

```
Sub File_open_1()
   Workbooks.Open Filename := "C:\Kaplan\Autos.xlsx"
End Sub
```

In the second example, the file and the path of the file are declared as a string, opened with input-only and without a request for the update of the links. It will be closed immediately after opening.

```
Sub File_open_2()
 Dim strFilename, strPath As String
 'Dim strFull As String
  strPath = "C:\Kaplan\"
  strFilename = "Autos.xlsx"
    Workbooks.Open Filename:=strPath & strFilename, ReadOnly:=True, UpdateLinks:=False
   'It can also be entered like this
   'Workbooks.Open Filename:=strFull , ReadOnly:=True, UpdateLinks:=False
Workbooks(strFilename).Close
End Sub
```

In our third example, we do not give the file name or the path to it. We choose them. This example was realized with FSO. To see all Excel files, we write the Excel extension with a placeholder "?", So (* .xls?).

Use the GetOpenFilename method to bring up the standard Open dialog box and select the Excel file. "VBA remembers" the file name with the path.

```
Sub File_open_3()
 Dim strDatei As String
   strDatei = Application.GetOpenFilename("MS Excel-Datei (*.xls?), *.xls?")
 Workbooks.Open Filename:=strDatei
End Sub
```

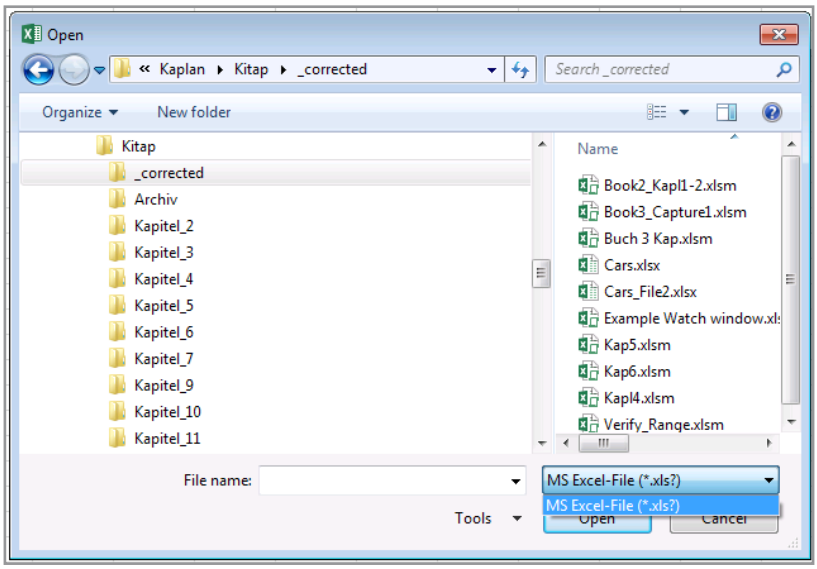


Figure 32: Standard dialog window of file open

We can also set our directory or the boot drive beforehand. This has the advantage of eliminating the need to switch directories. We can expand our example so that text files are also visible. Then our current and last example would look like this. In the chapter "open workbook" we have already looked at.

```
Sub File_open_4()
 Dim strFile As String
  ChDir "C:\Kaplan\"
  ChDrive "C:\"
   strFile = Application.GetOpenFilename("MS Excel-File (*.xls?), *.xls?, TextFile (*.txt), *.txt"
 Workbooks.Open Filename:=strFile
End Sub
```

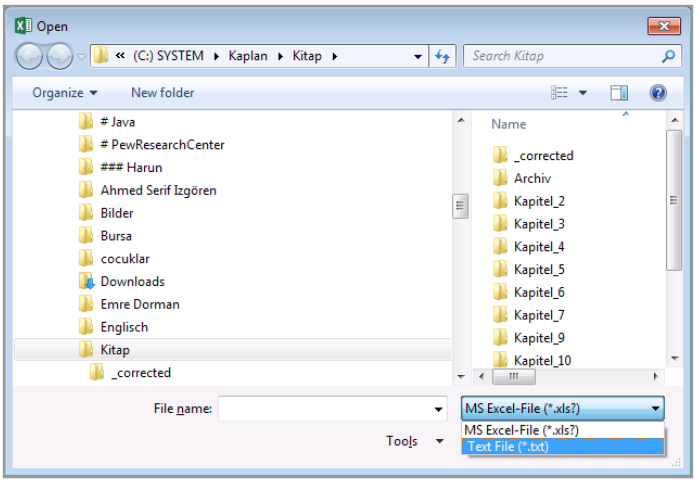


Figure 33: Standard dialog window of open with different data type

3.1.2 DELETE A FILE

This Kill-statement can be used to delete one or more files from a volume. It does not matter if this is an Excel file or another file type. It is eliminated.

The statement can also use the asterisk "*" as a wildcard for multiple characters and the question mark "?" For a single character to specify multiple files.

Our directory looks like this:

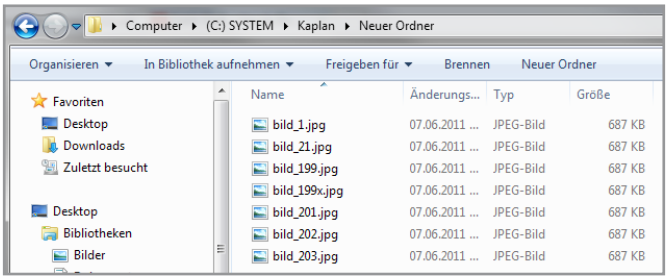


Figure 34: Directory of pictures

Now for our example. Here we have no presence control in it. We assume that the "file picture_1.gif" is in the directory "Harun". Otherwise we will receive a debug error.

```
Sub Delete_a_file()
   Kill ("C:\Kaplan\New Directory\picture_1.jpg")
End Sub
```

The next example is shown with a wildcard.

In this example, all image files starting with "2" are deleted.

```
Sub Delete_with_wildcard_1()
   Kill ("C:\ Kaplan\New Directory\picture_2*.jpg")
End Sub
```

This example irrevocably deletes all files in the directory.

```
Sub Delete_with_wildcard_2()
   Kill ("C:\ Kaplan\New Directory\*.*")
End Sub
```

In the next example all image files, "picture_201.jpg" to "picture_203.jpg" will be deleted. Remaining files remain unaffected.

```
Sub Delete_with_wildcard_3_for_a_character()
   Kill ("C:\ Kaplan\New Directory\picture_?0?.jpg")
End Sub
```

Next example with FSO:

```
Sub File_delete_with_FSO_1()
 CreateObject("Scripting.FileSystemObject").DeleteFile ("C:\kaplan\New Directory\picture_221
.jpg")
End Sub
```

3.1.3 RENAMING A FILE NAME

With the name statement we can rename a file, a directory, or a folder. This statement can also be used to move a file. This is nothing else than cutting and pasting a file.

Syntax looks like this:

Name "Source_Directory\Filename_old" As "Destination_Directory\Filename_new"

Our first example looks like this. Here we assume that the source file (Picture_1.jpg " exists.

After the second start we get an error message.

```
Sub Renamed_a_filename_1()
 Name ("C:\Kaplan\New Directory\picture_1.jpg") As ("C:\Kaplan\New Directory\picture_399
.jpg")
End Sub
```

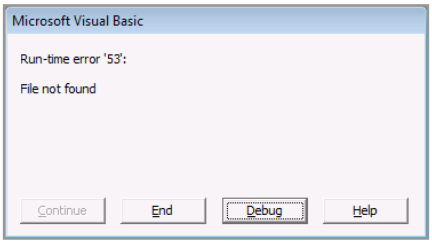


Figure 35: Error message after the second loop

In the next example we assume that the folder "New Directory" exists. The image file "picture_199.jpg" will be renamed "picture_001.jpg" and moved to the folder.

```
Sub Renamed_a_filename_2()
Name  ("C:\Kaplan\New Directory\picture_199.jpg") As _
        ("C:\Kaplan\New Directory\Ordner\picture_001.jpg")
End Sub
```

Well, it is all well and good. Now we will incorporate a presence control. For this we use the Dir function to help. This is the name of a file, a directory or a folder.

```
Sub Renamed_a_filename_3()
'Control of exist a source file
  If Dir("C:\Kaplan\New Directory\picture_1.jpg") = "" Then
    MsgBox "The file does not exist"
    Exit Sub
  Else
'Control of destination directory
    If Dir("C:\Kaplan\New Directory\picture_199.jpg") <> "" Then
      MsgBox "The file already exists”
    Else
'Renaming takes place here
      Name ("C:\Kaplan\New Directory\picture_1.jpg") As ("C:\Harun\picture_199.jpg")
    End If
  End If
End Sub
```

It may be that our directory or filenames are not as short as in our examples so our previous examples may look a bit better:

```
Sub Renamed_a_filename_4()
Dim strDirectory As String
Dim strSourcefile, strDestinationfile As String

strDirectory = "C:\Kaplan\New Directory\"
strSourcefile = "picture_1.jpg"
strDestinationfile = "picture_199.jpg"

  If Dir(strDirectory & strDestinationfile) = "" Then
    MsgBox "The file " & strSourcefile & " does not exist"
    Exit Sub
  Else
    If Dir(strDirectory & strDestinationfile) <> "" Then
      MsgBox "The file " & strDestinationfile & " already exist."
    Else
      Name (strDirectory & strDestinationfile) As (strDirectory & strSourcefile)
    End If
  End If
End Sub
```

3.1.4 COPY A FILE

One or more files are copied using the **FileCopy** statement. The original file is not lost, but stored in the same folder under different names or in another folder with the same name.

Syntax looks like this:

FileCopy “Source_directory\Files”, “Destination_directory\Files”

Our first example copies the file in the same folder under the new name.

```
Sub Copy_of_a_file_1()
Dim strSourcefile, strDestinationfile As String
  strSourcefile = "C:\Kaplan\New Directory\picture_199.jpg"
  strDestiantionfile = "C:\Kaplan\New Directory\picture_199x.jpg"
  FileCopy strSourcefile, strDestinationfile
End Sub
```

The second example copies the file with the same name one level down

```
Sub Copy_of_a_file 2()
Dim strSourcefile, strDestinationfile As String
 strSourcefile = "C:\Kaplan\New Directory\picture_199.jpg"
 strDestinationfile = "C:\Kaplan\New Directory\Ordner\picture_199.jpg"
 FileCopy strSourcefile, strDestinationfile
End Sub
```

In the introduction of this chapter I mentioned that the renaming is done by "Save As". We will check first the source file then the target file for the presence.

Syntax looks like this:

CopyFile "Source_directory\Files", "Destination_directory\Files"

Our next example is with CopyFile from File-System-Object (FSO). We use an asterisk (*) and the question mark (?) as a bookmark. Here, all files that start with "image" will be copied to the new folder "folder".

```
Sub Copy_of_a_file 3()
 Dim stroldpath, strnewpath As String
 Dim objFSO As Object
   stroldpath = "C:\Kaplan\New Directory\"
   strnewpath = "C:\Kaplan\New Directory\Ordner\"
    Set objFSO = CreateObject("Scripting.FileSystemObject")
      objFSO.CopyFile stroldpath & "picture*.*", strnewpath
    Set objFSO = Nothing
End Sub
```

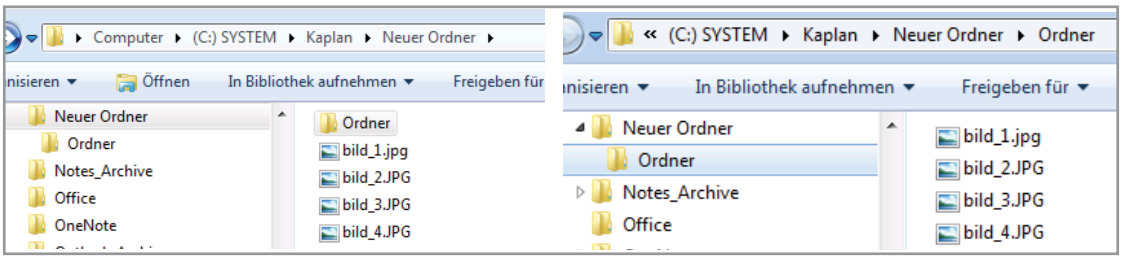


Figure 35: Content copied from "New Directory" in "Ordner"

3.1.5 MOVE A FILE

Moving a file from A to B is also possible through renaming. This is also done with the **MoveFile** statement. Moving means that a file is cut from a folder and pasted into a new folder.

An example of this looks like this:

```
Sub Move_a_file_1()
Dim strOldPath, strNewPath As String
Dim strSourcefile As String
Dim objActiveX As Object

strOldPath = "C:\Kaplan\"
strNewPath = "C:\Kaplan\Unterverzeichnis\"
strSourcefile = "picture_1.jpg"

Set objActiveX = CreateObject("Scripting.FileSystemObject")

  If Dir(strOldPath & strSourcefile) = "" Then 'Control of exist source file
    MsgBox "The file " & strSourcefile & " does not exist."
    Exit Sub
  Else
    If Dir(strPfadneu & strSourcefile) = "" Then 'Control of exist destination file
      MsgBox "The file " & strSourcefile & " is exist."
    Else
      objActiveX.MoveFile strOldPath & strSourcefile, strNewPath
    End If
  Set objActiveX = Nothing
  End If
End Sub
```

3.1.6 CHECK AN EXISTING FILE

This checks if a specific file exists.

```
Sub Contol_of_existing_a_file_1()
    If CreateObject("Scripting.FileSystemObject").FileExists _
                    ("C:\kaplan\New Directory\picture_221.jpg") = True Then
      MsgBox "Exist."
    Else
     MsgBox "Not exist."
   End If
End Sub
```

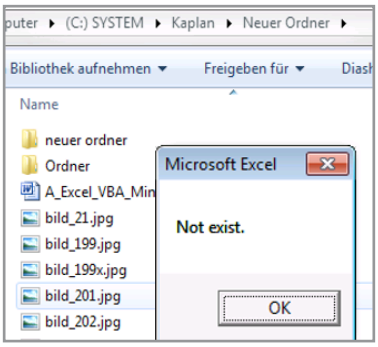


Figure 36: Check a file

3.1.7 DETERMINING A FILE NAME OR DIRECTORY NAME

In the current example, we will read out file names and subdirectory names of a specific folder. Read file names are entered in column A and directory names in column B.

We use the following methods:

- **Files** reads out the file names,
- **SubFolders** reads the directory names.

```
Sub Read_the_filenames_from_directory()
Dim objFSO As Object
Dim strInput As String
Dim strMsg As String
Dim i As Integer
Dim content As Object
Set objFSO = CreateObject("Scripting.FileSystemObject")
  strInput = "C:\Kaplan\"
    If Dir(strInput) = "" Then
      MsgBox "The path " & strInput & " does not exixt."
      Exit Sub
    End If
  i = 1
  For Each content In objFSO.GetFolder(strInput).Files 'Files
    Cells(i, 1).Value = content.Name
    i = i + 1
  Next
  i = 1
  For Each content In objFSO.GetFolder(strInput).SubFolders 'Dircetory
    Cells(i, 2).Value = content.Name
    i = i + 1
  Next
Set objFSO = Nothing
End Sub
```

	A	B
1	Autos.xlsx	Neuer Ordner
2	Dateien_Verzeichnis_suchen.xlsm	
3	Dateien_Verzeichnis_suchen.xlsx	
4	Funktionen.xlsm	
5	Rücknahme - Kopie (2).xlsm	
6	Rücknahme - Kopie.xlsm	
7	Rücknahme.xlsm	
8	Rücknahme.xlsx	
9	Rücknahme_2.xlsm	
10	Suchen_Ersetzen von Zellinhalten.xlsm	
11	SystemInformationen.xlsm	
12	Unsere txt-Datei.txt	
13	UserForm_NEU.xlsm	

Figure 37: Contend a directory

Here is an example to separate the file name and path.
If we somehow read this path "C:\Directory\Subdirectory\VBA_Themes\VBA.xlsm", it may be important to disconnect them. Only highlight the file name and the path:

```
Sub Cuting_filename_and_dirctoryname()
   Filename_with_path = "C:\Directory\Subdirectory\VBA_Themen\VBA.xlsm"
   Set Creob = CreateObject("Scripting.FileSystemObject")
      Only_filename = Creob.GetFileName(Filename_with_path)
      Only_path = Creob.GetParentFolderName(Filename_with_path)
      MsgBox Only_filename & vbLf & Only_path
  Set Creob = Nothing
End Sub
```

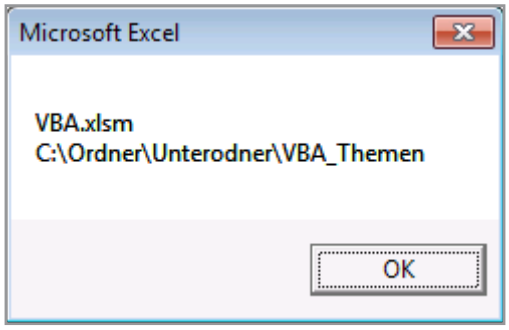


Figure 38: Result Cutting_filename_and_directoryname

3.1.8 READ THE PROPERTY FROM A FILE

The properties of a file are visible with a right click on the file name in Windows Explorer as shown in Figure.

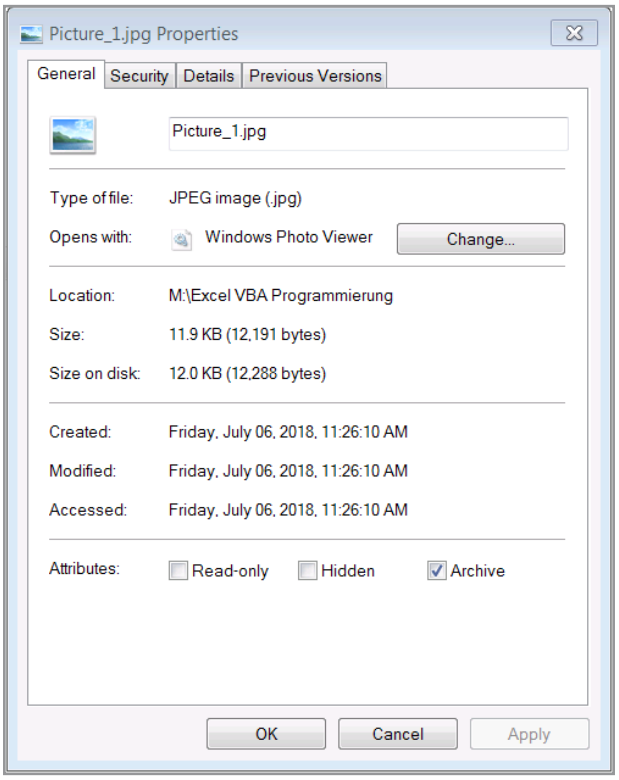


Figure 39: File Property

The information of a file can be searched with the command "**GetFile**". We have previously read in the file names and directories located there. We add this in order to attach the properties as a comment. Our result should look like shown below:

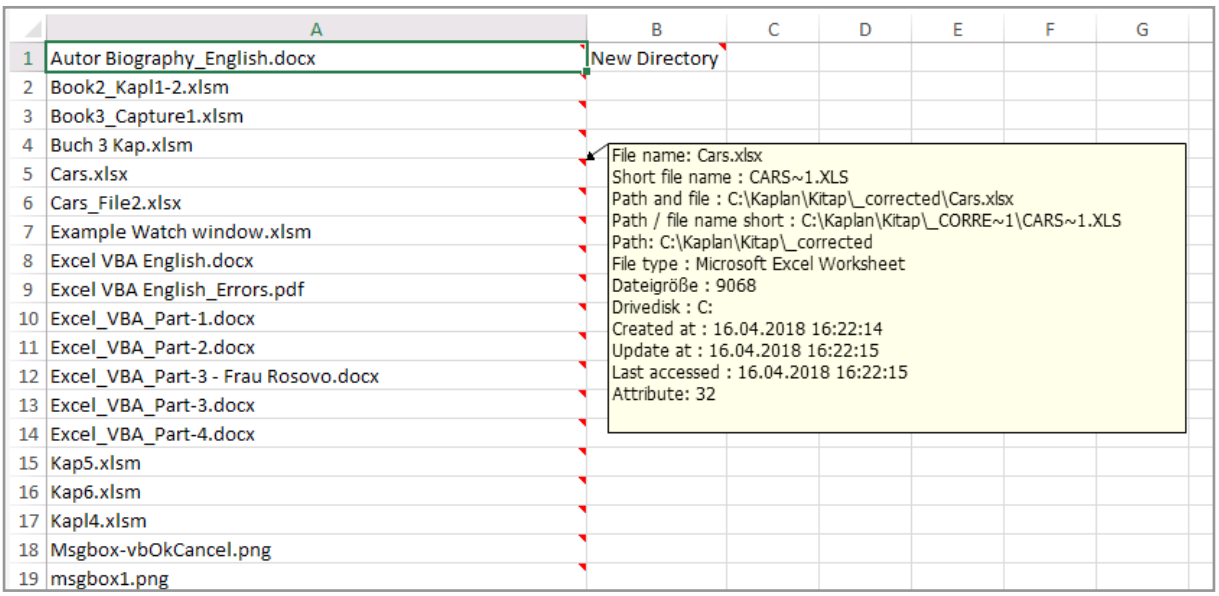


Figure 40: Listed directory content with file properties as comment

Here is our listing for them:

```
Sub Content_Directory_with_Eigenschaften()
Dim objFSO As Object
Dim strInput, strArt As String
Dim i As Integer
Dim objContent As Object
Dim strComment As String

Set objFSO = CreateObject("Scripting.FileSystemObject")

  strInput = InputBox("Enter Path ", Default:="C:\Harun\")
    If Dir(strInput) = "" Then
      MsgBox "The Path " & strInput & " does not exist.!"
      Exit Sub
    End If
  i = 1
  For Each objContent In objFSO.GetFolder(strInput).Files  'Files
 strArt = "File name: "
    With Cells(i, 1)
    .Value = objContent.Name
   GoSub Comment:
       With .AddComment
        .Shape.Height = 150
        .Shape.Width = 300
       End With
    .Comment.Text Text:=strComment
   End With
    i = i + 1
  Next
  i = 1
  For Each objContent In objFSO.GetFolder(strInput).SubFolders  'Directory
strArt = "Directory name : "
    With Cells(i, 2)
    .Value = objContent.Name
    GoSub Comment:
       With .AddComment
        .Shape.Height = 150
        .Shape.Width = 300
       End With
    .Comment.Text Text:=strComment
   End With
    i = i + 1
  Next
Set objFSO = Nothing
Exit Sub
Comment:
strComment = strArt & objContent.Name & vbNewLine & _
                  "Short file name : " & objContent.ShortName & vbNewLine & _
                  "Path and file : " & objContent.Path & vbNewLine & _
                  "Path / file name short : " & objContent.ShortPath & vbNewLine & _
                  "Path: " & objContent.ParentFolder & vbNewLine & _
                  "File type : " & objContent.Type & vbNewLine & _
                  "File size : " & objContent.Size & vbNewLine & _
                  "Drivedisk : " & objContent.Drive & vbNewLine & _
                  "Created at : " & objContent.DateCreated & vbNewLine & _
                  "Update at : " & objContent.DateLastModified & vbNewLine & _
                  "Last accessed : " & objContent.DateLastAccessed & vbNewLine & _
                  "Attribute: " & objContent.Attributes
Return
End Sub
```

Properties	Key
Full file name	.Name
Short file name	.ShortName
Path and full file name	.Path
Path and short file name	.ShortPath
Path	.ParentFolder
File type	.Type
File size	.Size
Drive	.Drive
Created at	.DateCreated
Updated at	.DateLastModified
Last accessed	.DateLastAccessed
Attribute	.Attributes

Table 3: Content of Object library

3.1.9 MAKE A DIRECTORY

Now we come to create or set-up a directory. We can not only create a directory via Explorer but also directly from Excel via VBA. This action is done with the statement **MkDir** "**Make Directory**". We write the directory to be created with the drive, if necessary, with the entire path.

In the example below, we will create a directory in the specified folder.

```
Sub Make_a_directory()
 Dim strNew_name As String
  strNew_name = "New Directory"
   MkDir ("C:\Kaplan\" & strNew_name)
End Sub
```

Figure 41: Created directory

If the same procedure goes through for the second time, we will get an error message because there is a directory with the same name. Now let's have a look at it and if not, create it.

```
Sub Make_a_directory_with_control()
 Dim strNew_Name As String
  strNew_Name = "New Directory"
   If Dir(("C:\Kaplan\" & strNew_Name), vbDirectory) = "" Then
     MkDir ("C:\Kaplan\" & strNew_Name)
   Else
     MsgBox "The directory " & Chr(10) & "'" & strNew_Name & "'" & vbLf & " is exist."
   End If
End Sub
```

Now we'll create a directory with the contents of a cell and the creation date.
Now for our example: In cell "A1" of "Sheet1" is the word "Apple".

Our directory should then be "Apple_Date_of_create".

The VBA listing and the result look like this:

```
Sub Make_a_directory_with_from_cellvalue()
 Dim strNew_Name As String
  strNew_Name = Worksheets("Sheet1").Range("A1").Value & "_" & Date
   If Dir(("C:\Kaplan\Test\" & strNew_Name), vbDirectory) = "" Then
     MkDir ("C:\Kaplan\Test\" & strNew_Name)
   Else
     MsgBox "The directory " & Chr(10) & "'" & strNew_Name & "'" & vbLf & " is exist."
   End If
End Sub
```

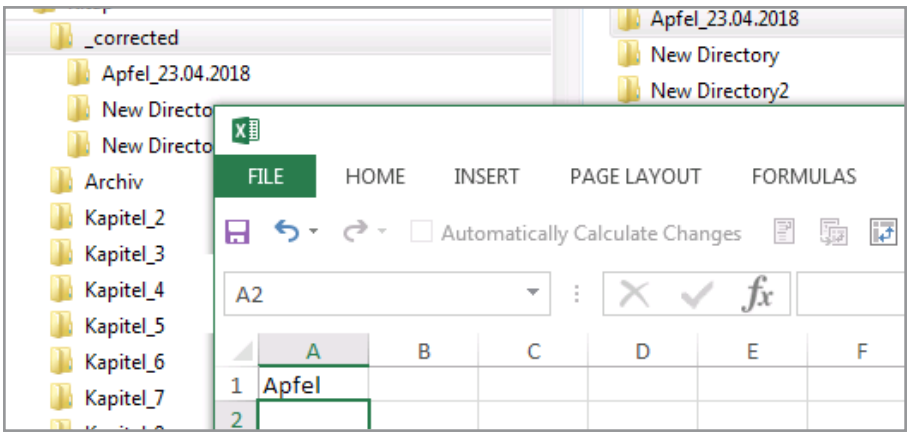


Figure 42: Result of Make_a_directory_with_from_cellvalue

3.1.10 CHANGE A DIRECTORY

We will now go to the directory that we created earlier. This process is then implemented with the "**ChDir Change Directory**" statement.

```
Sub Chage_a_directoy()
ChDir („C:\Kaplan\Test\Apple_15.02.2018\")
End Sub
```

3.1.11 DELETE A DIRECTORY

All created directories can also be deleted. This process can be done in two ways:

1. **RmDir** „**Remove Directory**",
2. **DeleteFolder**-Objekt

```
Sub Delete_a_directory()
 RmDir („C:\Kaplan\Test\Apfel_15.02.2018\")
End Sub

Sub Delete_a_directory_FSO_1()
 CreateObject("Scripting.filesystemobject").DeleteFolder („C:\Kaplan\Test\Apfel_15.02.2018\")
End Sub
```

3.1.12 CHECK EXISTING DIRECTORIES

Now we check to see if a folder exists:

```
Sub Control_Exist_of_directory_1()
   If CreateObject("Scripting.FileSystemObject").FolderExists _
                    ("C:\kaplan\New Directory") = True    Then
    MsgBox "Exist."
  Else
   MsgBox "Does not exist."
  End If
End Sub
```

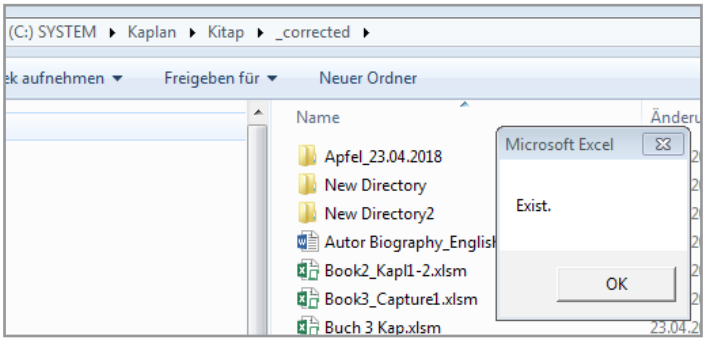


Figure 43: Result directory of existing

3.1.13 PROPERTY FROM A DRIVER

We now search all the available drive properties in one go and summarize them in a comment. We will do this the same way we did when determining the properties of a file.

Our VBA listing looks like this:

```
Sub Driver_Property ()
Dim objFSO, objDrive As Object
Dim i As Integer
Dim strComment As String

Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objDrive = objFSO.GetDrive("C:\")
  i = 1
    With Cells(i, 5)
      .Value = objDrive
    GoSub Comment:
       With .AddComment
        .Shape.Height = 150
        .Shape.Width = 300
       End With
    .Comment.Text Text:=strComment
   End With
    i = i + 1
Set objFSO = Nothing
Exit Sub
Comment:
strComment = "Drive letter: " & objDrive & vbNewLine & _
                  "Available space: " & objDrive.AvailableSpace & vbNewLine & _
                  "Free space: " & objDrive.FreeSpace & vbNewLine & _
                  "Drive type: " & objDrive.DriveType & vbNewLine & _
                  "File system : " & objDrive.FileSystem & vbNewLine & _
                  "Ready : " & objDrive.IsReady & vbNewLine & _
                  "Path : " & objDrive.Path & vbNewLine & _
                  "Root directory of drive : " & objDrive.RootFolder & vbNewLine & _
                  "Share name: " & objDrive.ShareName & vbNewLine & _
                  "Total size of drive: " & objDrive.TotalSize & vbNewLine & _
                  "Volume name: " & objDrive.VolumeName
Return
End Sub
```

The result of them:

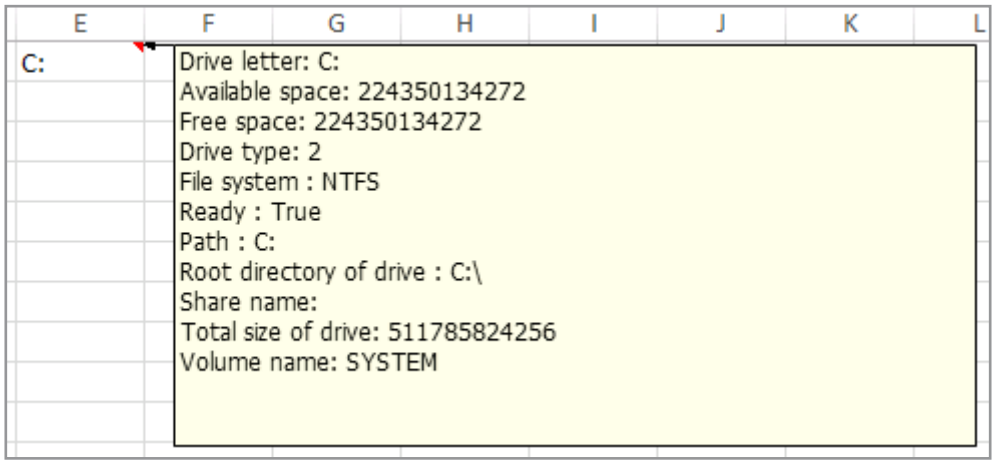


Figure 44: Result Driver_Property

3.1.14 READ THE SYSTEM INFORMATION

So, slowly but surely, we are approaching the end of this chapter. We will now learn to read the system information. We can do that with the Environ method.

The next example lists all readable system information. VBA code is run through until all information has been read out. The result is listed in the table from column E onwards.

```
Sub Read_the_Systeminformation()
 Dim i As Integer
  i = 1
   Do While Environ(i) <> ""
    Cells(i, 6) = i & " .) " & Environ(i)
    i = i + 1
   Loop
End Sub
```

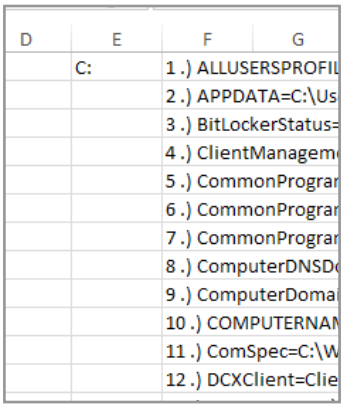


Figure 45: Part of the result

From this information the UserName is most important (for me). For single queries, the element I want to have is entered in quotes.

If we specifically search for “username”, we will write it this way: Environ (“UserName”).

```
Sub Read_Username()
  Dim strName As String
   strName = Environ("UserName")
    If strName = "Mustermann" Then
      MsgBox "You can continue."
   Else
    Exit Sub
  End If
End Sub
```

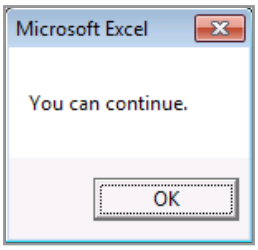


Figure 46: result, if user is right.

4 CREATING A CHART

An important part of Excel is the creation of diagrams. I will discuss two options here:

- Create a new diagram / chart area
- Display existing chart area with new data.

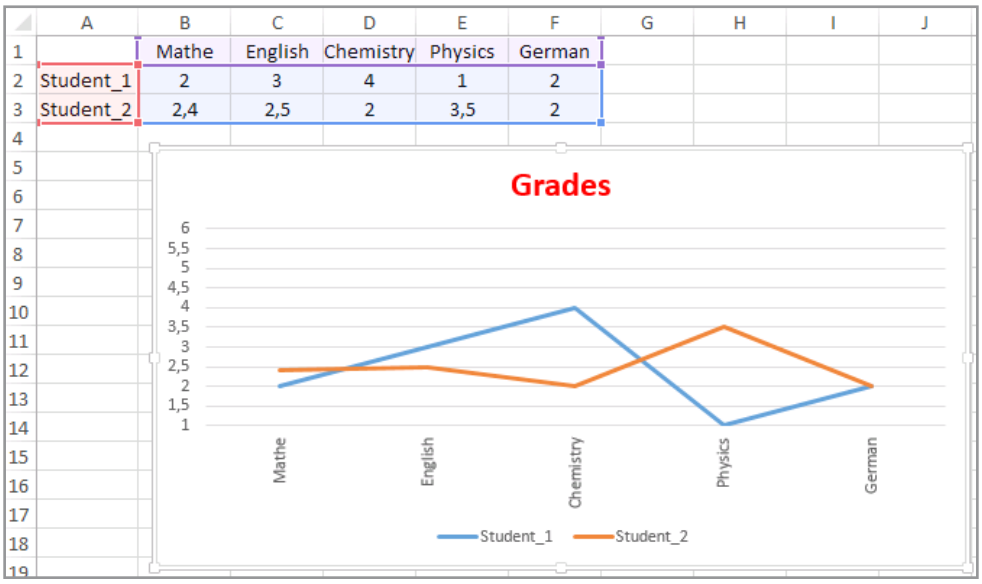
A diagram consists of the following elements:

- Axes
- Data series
- Legends
- Labeling

Here we will set up a simple new diagram. The listing is actually self-explanatory. I have kept it as short as possible. In the VBA editor, you can open the help text for each command and look at other supplementary options.

```
Sub Charte_1()
    Range("A1:F3").Select
    ActiveSheet.Shapes.AddChart2(227, xlLine).Select  'Art of chart Line
    ActiveChart.SetSourceData Source:=Range("Sheet1!$A$1:$F$3") 'Source data
'X-Axes setting
    With ActiveChart.Axes(xlValue)
      .Select
      .MinimumScale = 1
      .MaximumScale = 6
    End With
'Y-Axes setting
    With ActiveChart.Axes(xlCategory)
      .Select
      .CategoryType = xlAutomatic
    End With
'Label for Y-Axes
    Selection.TickLabels.Orientation = xlUpward
'Titel insert
    With ActiveChart.ChartTitle
      .Select
      .Text = "Grades"
    End With
'Title formated
    With Selection.Format.TextFrame2.TextRange.Font
      .Size = 18
      .Bold = msoTrue
        With .Fill
          .Visible = msoTrue
          .ForeColor.RGB = RGB(255, 0, 0)
          .Transparency = 0
          .Solid
        End With
    End With
End Sub
```

The data area and the result looks like this:

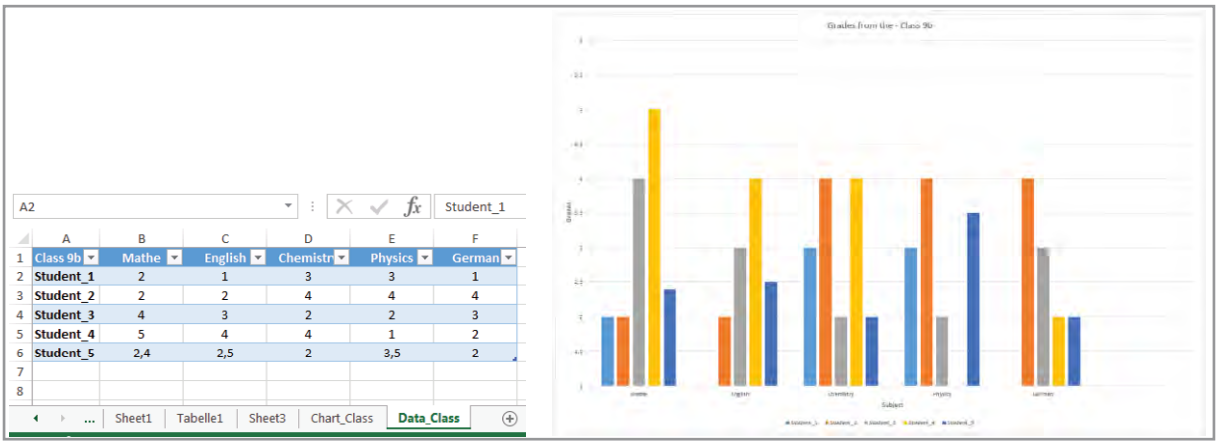


	A	B	C	D	E	F
1		Mathe	English	Chemistry	Physics	German
2	Student_1	2	3	4	1	2
3	Student_2	2,4	2,5	2	3,5	2

Figure 47: Result of Chart_1

Now we fill our diagram with new data. This option is useful if the diagram is to be created with different values from a database. I find this kind of procedure even more practical. For example, when I have a chart sheet it will always be updated with new data (single data series).

This is what the raw data and graph looks like:



	A	B	C	D	E	F
1	Class 9b	Mathe	English	Chemistr	Physics	German
2	Student_1	2	1	3	3	1
3	Student_2	2	2	4	4	4
4	Student_3	4	3	2	2	3
5	Student_4	5	4	4	1	2
6	Student_5	2,4	2,5	2	3,5	2

Figure 48: Result of Class_Grades

Our listing:

```
Sub Class_Grades()
Dim i As Integer
On Error Resume Next
Application.ScreenUpdating = False
Worksheets("Data_Class").Select
Title = "Grades from the - " & Range("A1").Value
Class = Range("A1").Value
Sheets("Chart_Class").Select
ActiveChart.ChartArea.Select
ActiveChart.PlotArea.Select
'Count of data series in chart
CountDataSeries = ActiveChart.SeriesCollection.Count
'Delete of data series
For DataSeries = CountDataSeries To 1 Step -1
   ActiveChart.SeriesCollection(DataSeries).Delete
Next DataSeries
Worksheets("Data_Class").Select
Range("A2").Select
Worksheets("Data_Class").Select
c = Cells(Rows.Count, 1).End(xlUp).Row
   Select Case Class
      Case "Class 9a", "Class 9b", "Class 9c"
         Min = 1
         Max = 6
         'Min2 = 1 'for second axes
         'Max2 = 3 'for second exes
      Case "Class 10a"
         Min = 1
         Max = 5
         'Min2 = 1
         'Max2 = 6
      Case "Class 10b", "Class 10c"
         Min = 2
         Max = 3
         'Min2 = 3
         'Max2 = 6
   End Select
i = 1
  For g = 2 To c
         Sheets("Chart_Class").Select
         With ActiveChart
            .ChartArea.Select
            .PlotArea.Select
            .SeriesCollection.NewSeries
            .FullSeriesCollection(i).Name = "= Data_Class 9b!$A$" & g
            .FullSeriesCollection(i).XValues = "= Data_Class 9b!$B$1:$G$1"
            .FullSeriesCollection(i).Values = "= Data_Class 9b!$B$" & g & ":$G$" & g
         End With
         With ActiveChart
            .Axes(xlValue).MinimumScale = Min
            .Axes(xlValue).MaximumScale = Max
         End With
' If necessary, scope for second axis inserted here!
'
         i = i + 1
      Next g
      ActiveChart.ChartArea.Select
' Chart title
      With ActiveChart
```

```
        .HasTitle=True
        .ChartTitle.Select
End With
        Selection.Caption = Überschrift
'Label of Y-Axes
        With ActiveChart.Axes(xlValue, xlPrimary)
            .HasTitle = True
            .AxisTitle.Text = "Grades"
        End With
'Label of X-Axes
        With ActiveChart.Axes(xlCategory, xlPrimary)
            .HasTitle = True
            .AxisTitle.Text = "Subject"
        End With
Worksheets("Data_Class").Select
End Sub


'Formating second axes

                With ActiveChart
                    .ChartArea.Select
                    .FullSeriesCollection(i).Select
                    .SeriesCollection(i).AxisGroup = 2
                    .SeriesCollection(i).Select
                    .Axes(xlValue, xlSecondary).MinimumScale = Min2
                    .Axes(xlValue, xlSecondary).MaximumScale = Max2
                End With
            End If
```

Hereby we can query the name of the data series:

- ⇨ DataSeries = ActiveChart.FullSeriesCollection(i).Name
- ⇨ The indes “i” indicates the data series.

Name of the third data series is determined in this way:

- ⇨ DataSeries_3 = ActiveChart.FullSeriesCollection(3).Name

5 CREATING OWN YOUR MENU

Now we will create our own menu and attach it to the menu bar. Any VBA particle beginning with “Sub” and ending with “End Sub” can be displayed as a menu.

These menu entries are then added under “Add-ins”.

Now we can see what our menu should look like as well as the VBA-Code:

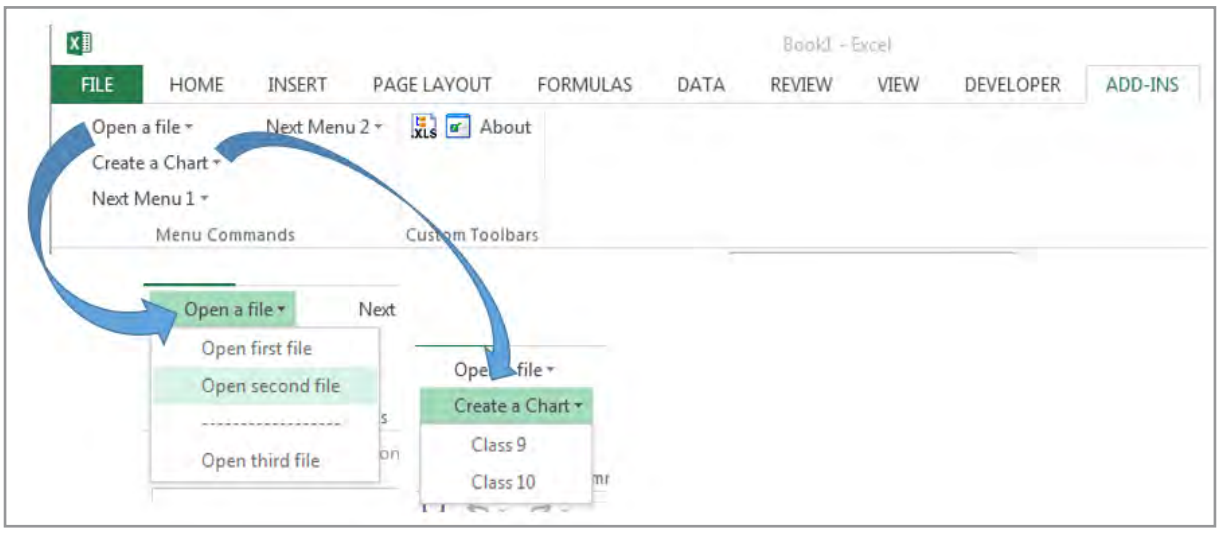


Figure 49: So soll unser Menü aussehen

Before we go into the VBA code, we will look at a couple of rules:

If we want to run our menu right at the start of the Excel file, we enter the following lines in the “This workbook”:

```
Private Sub workbook_open()
 Call MyMenue
End Sub
```

Also, when exiting the Excel file our menu should be deleted. Otherwise we have many menu entries under “Add-ins”.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
On Error Resume Next
   Application.CommandBars("Worksheet Menu Bar").Controls("Open a file").Delete
   Application.CommandBars("Worksheet Menu Bar").Controls("Create a chart").Delete
   Application.CommandBars("Worksheet Menu Bar").Controls("Next Menü1").Delete
   Application.CommandBars("Worksheet Menu Bar").Controls("Next Menü2").Delete
End Sub
```

Here is a short explanation of the VBA code:

Letter: It indicates the menu group or menu level,
Letter**0**.Caption The number after letter indicates topmost menu item
Letter0.**Caption** menu caption
Letter1.**OnAction** macro to be executed

a1.OnAction="Modulename.Makro_to_be_executed"

Example:
Modul1 contains a macro „Open_File". This macro is called as follows:
a1.OnAction="Modul1.Open_File"

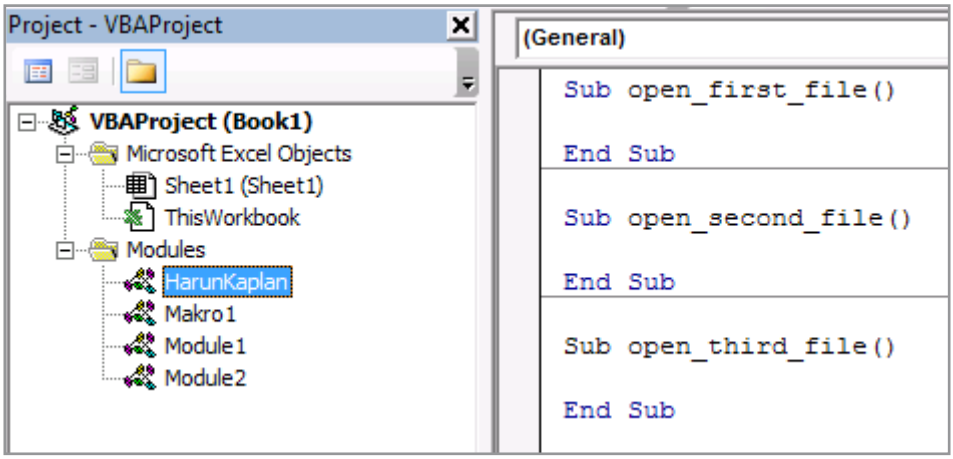


Figure 50: Structure of Modules

Here is the listing:

```
Sub MyMenue()
On Error GoTo myMenu

myMenu:
On Error GoTo 0
Set a0 = Application.CommandBars.ActiveMenuBar.Controls.Add(msoControlPopup)
   a0.Caption = "Open a file"
        Set a1 = a0.Controls.Add(msoControlButton)
           a1.Caption = "Open first file"
           a1.OnAction = "HarunKaplan.Open_first_file"
        Set a2 = a0.Controls.Add(msoControlButton)
           a2.Caption = "Open second file"
           a2.OnAction = "HarunKaplan.Open_second_file"
        Set a3 = a0.Controls.Add(msoControlButton)
           a3.Caption = "------------------"
        Set a4 = a0.Controls.Add(msoControlButton)
           a4.Caption = "Open third file"
           a4.OnAction = "HarunKaplan.Open_third_file"
'
Set b0 = Application.CommandBars.ActiveMenuBar.Controls.Add(msoControlPopup)
   b0.Caption = "Create a Chart"
        Set b1 = b0.Controls.Add(msoControlButton)
           b1.Caption = "Class 9"
           b1.OnAction = "Makro1.Grades_Class9"
        Set b2 = b0.Controls.Add(msoControlButton)
           b2.Caption = "Class 10"
           b2.OnAction = "Makro1.Grades_Class10"

'
Set c0 = Application.CommandBars.ActiveMenuBar.Controls.Add(msoControlPopup)
   c0.Caption = "Next Menu1"
        Set c1 = c0.Controls.Add(msoControlButton)
           c1.Caption = "Next Menu1"
           c1.OnAction = "Module2.Test1"
'
Set d0 = Application.CommandBars.ActiveMenuBar.Controls.Add(msoControlPopup)
   d0.Caption = "Next Men2"
        Set d1 = d0.Controls.Add(msoControlButton)
           d1.Caption = "Next Menu2"
           d1.OnAction = "Module1.Test2"
End Sub
```

BIBLIOGRAPHY

Online Microsoft Developer Network:

https://msdn.microsoft.com/de-de/vba/office-vba-reference