How to work with text, number, dates in VBA

Created by:	Marek Prazak	**Reviewed by:**	Hari Krish	**Create Date:**	14/04/2019

Contents

1.0 Purpose

- This lesson will show you how to work with text, numbers, dates in VBA, features like these will allow you to keep or adjust formatting as you wish without need of doing any changes to excels range formatting.
- The lesson should take approximately 40 min for completion.
- As the lessons are simplified to allow you to grasp the basics of VBA functionality, do not despair if you want understand all on your first walkthrough, we highly encourage you to use additional sources provided by us, to go online and research more in depth and mainly try out all examples highlighted in this lesson.

2.0 Definition

- Text function in VBA are very similar to Excel functions (Join, Split, Count, etc.).
- String is name of a variable that can hold text (if you store number in this variable, you will lose the option to count with that num. and you'll be forced to use conversion of type)
- Dates in VBA can be used as the Input, Output or loading Day, Month, Year etc.
- Number functions in VBA, same as Dates and Text, can be used as the Input, Output and have similar build-in functions as Excel functions. (count, sum, divide, etc.)
- In excel we will meet term worksheet and sheet. Even though thus far we've referred to both indifferently, Worksheet is a standard tab that you will see, if you open new workbook, Sheet on the other hand are all tabs that are to be created in excel (Chartsheet, Worksheet, etc.)
- Events are actions performed by users which trigger Excel VBA to execute code.

3.0 Overview

3.1 String Manipulation

VBA provides a large number of built-in String and Numeric functions that can be used in your code. The most popular **built-in VBA String** functions are listed below. You can use this function for any number of scenarios such as, correcting formatting with **Trim** to remove unnecessary spaces from hand typed words or converting the text to lower(**LCase**) or upper case(**Ucase**) to get unified look of logins. Example of how to use split with **LEFT, RIGHT, MID** you'll see later on. As you'll notice by trying yourself each Function has a specific syntax (More info on syntax of formulas in additional knowledge section).

VBA Text Functions

Format	Applies a format to an expression and returns the result as a string.
InStr	Returns the position of a substring within a string.
InStrRev	Returns the position of a substring within a string, searching from right to left.
Left	Returns a substring from the start of a supplied string.
Len	Returns the length of a supplied string.
LCase	Converts a supplied string to lower case text.
LTrim	Removes leading spaces from a supplied string.
Mid	Returns a substring from the middle of a supplied string.
Replace	Replaces a substring within a supplied text string.
Right	Returns a substring from the end of a supplied string.
RTrim	Removes trailing spaces from a supplied string.
Space	Creates a string consisting of a specified number of spaces.
StrComp	Compares two strings and returns an integer representing the result of the comparison.
StrConv	Converts a string into a specified format.
String	Creates a string consisting of a number of repeated characters.
StrReverse	Reverses a supplied string.
Trim	Removes leading and trailing spaces from a supplied string.
UCase	Converts a supplied string to upper case text.

3.1.1SEPARATE TEXT

- Below you can see 3 examples of functions used to Separate a text in VBA.

Mid(*Input Text, Start Num, No. of Chars* **)**

Input Text – Original Text from which we need to extract characters

Start Num – Position from where in the Input Text, Characters has to be extracted

No. of Chars – An optional argument representing the length of the substring. If the [No. of Chars] argument is omitted, the Mid function returns all characters from the Start position to the end of the string.

```
Sub text()

    Dim res As String

    res = Mid("John Michael Smith", 6, 7)

    MsgBox res

End Sub
```

- In above example we've declared res as string variable and later on assigned text (John Michael Smith) to it. On this text or more precisely name we've applied string function **MID**, where same as in excel functions/formulas we are splitting the text from 6th character and following 7 characters. It is also possible to change the text or name for string variable that would hold text that you want to split(below). This will allow you to use looping methods to split whole range of names or texts.

```
Sub text()

    Dim name As String, res As String

    name = "John Michael Smith"

    res = Mid(name, 6, 7)

    MsgBox res

End Sub
```

Right(*Input Text, No. of Chars***)**

Input Text – Original Text from which we need to extract characters

No. of Chars – Count of characters that has to extracted from the Original Text

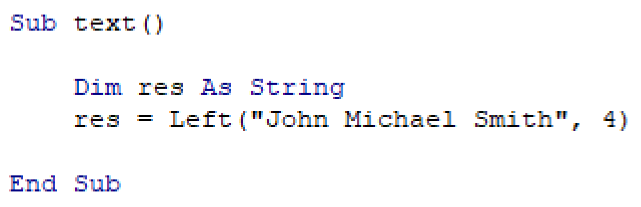
```
Sub text()

    Dim res As String
    res = Right("John Michael Smith", 5)

End Sub
```

- Similar to **MID** this function will allows you to split name from the 5th character from the right till the end (*Smith*).

LEFT(*Input Text, No. of Chars***)**

Input Text – Original Text from which we need to extract characters

No. of Chars – Count of characters that has to extracted from the Original Text

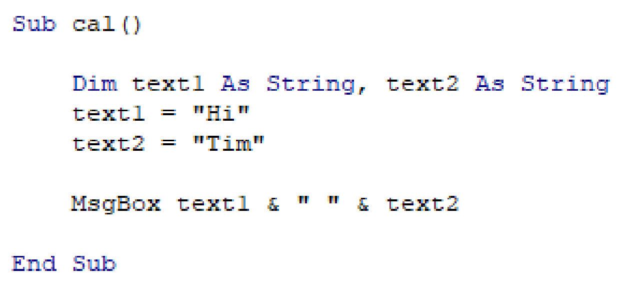

```
Sub text()

    Dim res As String
    res = Left("John Michael Smith", 4)

End Sub
```

- As you might have guessed already this function will allow you to split the text from 4th character from the left till the beginning (*John*)

Note: *Everything in declared in variable of string is considered text (special characters, numbers, even break/space).*

3.1.2 Concatenate Text

- Unlike in Excel there is not one specific way to concatenate, instead we use **& " " &**. You can concatenate Text, Numbers, Hyperlinks, etc.


```
Sub cal()

    Dim text1 As String, text2 As String
    text1 = "Hi"
    text2 = "Tim"

    MsgBox text1 & " " & text2

End Sub
```

- In above example we've declared two string variables (text1, text2) and assigned text to them. Following that we've used concatenate method to add text together. With a space or custom text in the middle.

```
'Hi Tim
MsgBox text1 & " " & text2

'Hi my best friend Tim
MsgBox text1 & " my best friend " & text2
```

3.2 Numbers in VBA

The most popular **built-in VBA Numeric functions** are listed below. VBA allows you to use similar functions to those of excel. By implementing them into your code you can create custom formulas that will do complex calculation as a result without having to worry about users accidentally damaging the nested formulas (formula in a formula in a formula) that has been created

VBA Math & Trig Functions

Abs	Returns the absolute value of a number.
Atn	Calculates the arctangent of a supplied number.
Cos	Calculates the cosine of a supplied angle.
Exp	Calculates the value of ex for a supplied value of x.
Fix	Truncates a number to an integer (rounding negative numbers towards zero).
Int	Returns the integer portion of a number (rounding negative numbers away from zero).
Log	Calculates the natural logarithm of a supplied number.
Rnd	Generates a random number between 0 and 1.
Round	Rounds a number to a specified number of decimal places.
Sgn	Returns an integer representing the arithmetic sign of a number.
Sin	Calculates the sine of a supplied angle.
Tan	Calculates the tangent of a supplied angle.
Sqr	Returns the square root of a number.

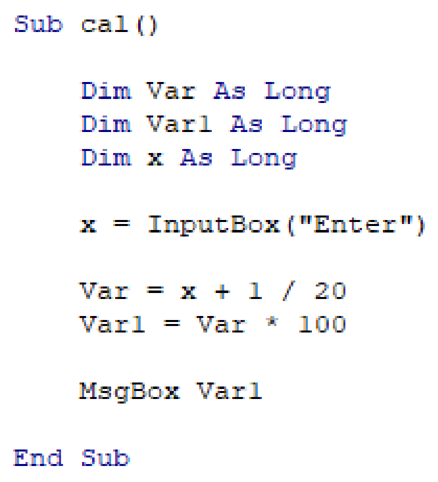
3.2.1 Declaring Numbers

- Most often you will use numbers in VBA as an Input to refer to a start point or the end point using Input box.
- To use number whether as an input or for a calculation, it is best practice to declare the numbers (less. 5) as correct type of variable.

  Integer – Can hold values between -32,768 to 32,767

  Long – Can hold values between -2,147,483,648 and 2,147,483,647

- As you've seen in the third lesson it is also possible to use VBA as a calculator with complexity depending on what you might need. Most of mathematical functions are available in VBA.



```
Sub cal()

    Dim Var As Long
    Dim Var1 As Long
    Dim x As Long

    x = InputBox("Enter")

    Var = x + 1 / 20
    Var1 = Var * 100

    MsgBox Var1

End Sub
```

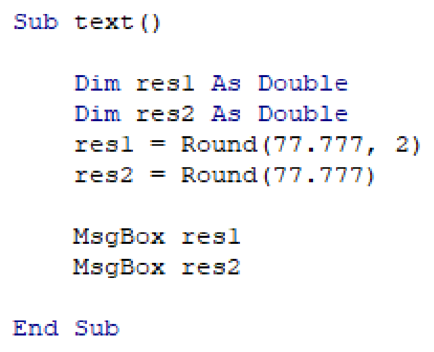
- In above example, if the user enters number 30 to the input box, this will be stored as x, so now x is equal to 30, then variable named Var will be calculated as 30 + 1/20, which is 1.5 and then a variable named Var1 will be calculated as 1.5 x 100, which is 150 and that number will appear in the Message box when the program is run.

3.2.2ROUND NUMBERS

- The VBA Round function rounds a number to a specified number of decimal places. This function is very useful if you're getting numbers with large number of decimal places or in case that you want to get only full number, you can see it's use quiet often when calculating salary/pay rate amounts.

  Round(*Number, NumDigitsAfterDecimal* **)**

  > **Number** - The number that you want to round.
  > **NumDigitsAfterDecimal** - An optional positive integer specifying the number of decimal places that you want to round to.



```
Sub text()

    Dim res1 As Double
    Dim res2 As Double
    res1 = Round(77.777, 2)
    res2 = Round(77.777)

    MsgBox res1
    MsgBox res2

End Sub
```

  - In above example we've declared two variables of type Double(decimals) and later on assigned them a value that has been rounded. As you might notice when you run the macro, res1 will be mathematically rounded to 2 decimals (77.78), on the other hand without specified number of decimals it will be mathematically rounded to no decimals (78).

3.3 DATE & TIME FUNCTIONS

The most popular **built-in VBA Date and Time functions** are listed below. As you might have noticed during your time of working with excel, users tend to use incorrect format of dates which will make macros incorrect or simply disable some functionality of a tracker due to invalid format. In other cases, you might want to get only specific information from date and you're forced to do either manual steps or unnecessary number of extra columns as side calculation (**Weekday**, **DateAdd**, etc.). For these moments you can use Date and Time functions.

VBA Date & Time Functions

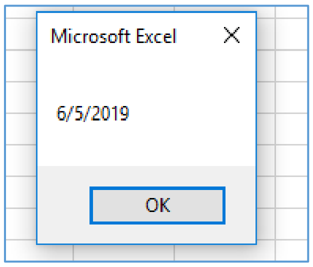
Date	Returns the current date.
DateAdd	Adds a time interval to a date and/or time.
DateDiff	Returns the number of intervals between two dates and/or times.
DatePart	Returns a part (day, month, year, etc.) of a supplied date/time.
DateSerial	Returns a Date from a supplied year, month and day number.
DateValue	Returns a Date from a String representation of a date/time.
Day	Returns the day number (from 1 to 31) of a supplied date.
Hour	Returns the hour component of a supplied time.
Minute	Returns the minute component of a supplied time.
Month	Returns the month number (from 1 to 12) of a supplied date.
MonthName	Returns the month name for a supplied month number (from 1 to 12).
Now	Returns the current date and time.
Second	Returns the second component of a supplied time.
Time	Returns the current time.
Timer	Returns the number of seconds that have elapsed since midnight.
TimeSerial	Returns a Time from a supplied hour, minute and second.
TimeValue	Returns a Time from a String representation of a date/time.
Weekday	Returns an integer (from 1 to 7), representing the weekday of a supplied date.
WeekdayName	Returns the weekday name for a supplied integer (from 1 to 7).
Year	Returns the year of a supplied date.

3.3.1 DATE FUNCTION

- simply returns the current date. The function takes **no parameters** and therefore, its syntax is:

Date( )

```
Sub Mydate()

    ' Store the current date in the variable currDate
    Dim currDate As Date
    currDate = Date

    MsgBox currDate

End Sub
```



```
6/5/2019
```

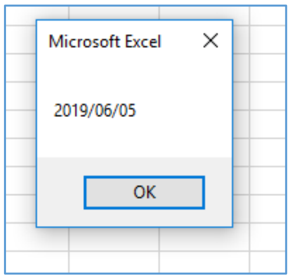
Note: *that date will use your computers formatting. To ensure preferred formatting, use **Format function**.*

Format(#17/04/2004#**,** "Short Date"**)**
Result: '17/04/2004'

Format(#17/04/2004#**,** "Long Date"**)**
Result: 'April 17, 2004'

Format(#17/04/2004#**,** "yyyy/mm/dd"**)**
Result: '2004/04/17'

```
Sub Mydate()

    Dim currDate As Date
    currDate = Date

    MsgBox Format(currDate, "yyyy/mm/dd")

End Sub
```



```
2019/06/05
```

3.3.2DATEADD FUNCTION

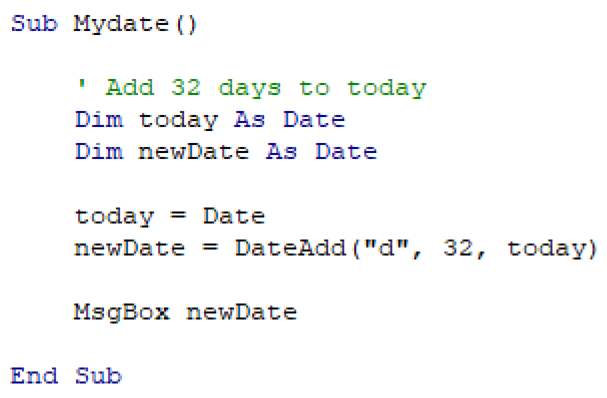
- adds a time interval to a supplied date and/or time, and returns the resulting date/time. The syntax of the **DateAdd** function is:

Dateadd(*Interval, Number, Date* **)**

Interval - A string specifying the interval to be used. This can have any of the following values:

"d"	-	Days
"h"	-	Hours
"n"	-	Minutes
"m"	-	Months
"q"	-	Quarters (of a Year)
"s"	-	Seconds
"ww"	-	Weeks
"yyyy"	-	Years

Number - The number of intervals to add to the specified Date.
Date - The original date/time that you want to add the specified number of intervals to.



```
Sub Mydate()

    ' Add 32 days to today
    Dim today As Date
    Dim newDate As Date

    today = Date
    newDate = DateAdd("d", 32, today)

    MsgBox newDate

End Sub
```

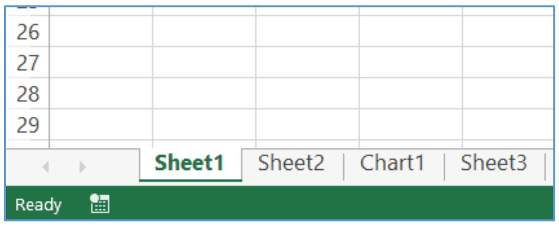
3.4 WORKBOOK IN VBA

- When working with excel you won’t be able to avoid situation where you have your program manipulating multiple excel workbooks or sheets at the same time. To make sure that your macro is doing exactly what you want, you’ll have to call/name concerned workbooks/sheets.

3.4.1 Worksheet Name

- There are three ways to refer to worksheet in VBA.
  - **Worksheet Name** – The easiest way to refer to a worksheet is as its reference name, that has been assigned to the sheet by a user (visible name of sheet: Sheet1, Sheet2, etc.).
    - Drawback of this technique is that, if the name of the sheet changes the VBA has to be manually adjusted

```
Sub ActivateSheet()

    Worksheets("Sheet2").Activate

    'or

    Sheets("Sheet2").Activate

End Sub
```

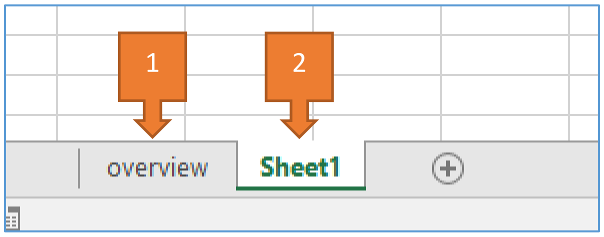


    - Activate parameter will allow you to highlight(select) concerned worksheet.

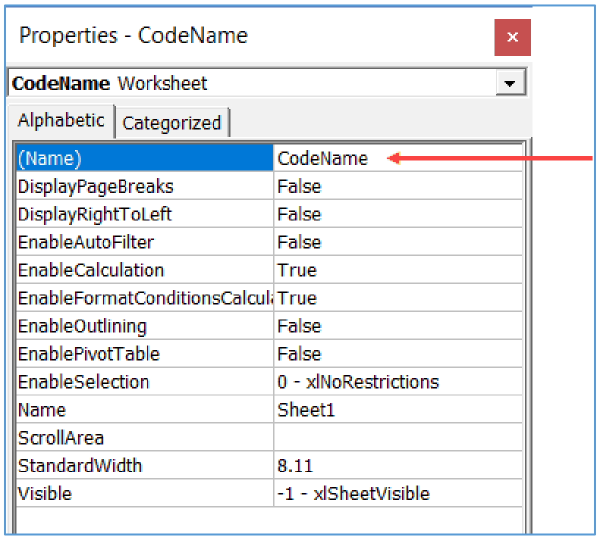
  - **Index Number** – In case that you do not know the exact name of the worksheet you can refer to it by its order number.

```
Sub ActivateSheet()

    Worksheets(2).Activate

End Sub
```

    - As the overview has been created first it is considered 1 sheet.



- **Worksheet Code Name** - you can use the code name of the worksheet (instead of the regular name that we have been using so far). A code name can be assigned in the VB Editor and doesn't change when you change the name of the sheet from the worksheet area.
  - To adjust Code name, navigate to Properties in VBA editor.

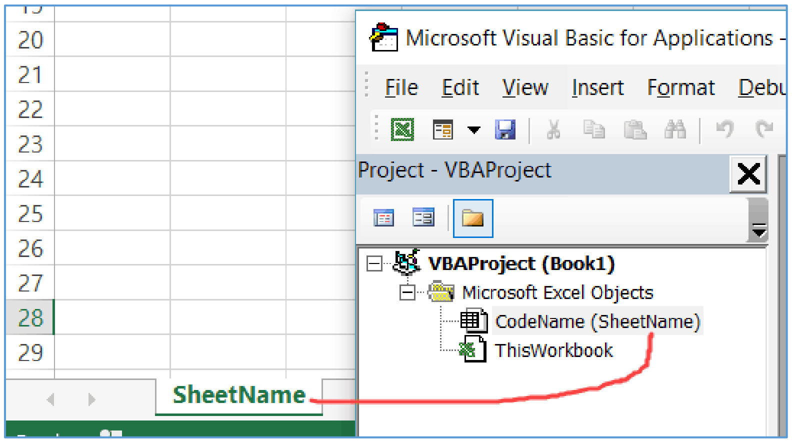


Properties - CodeName

CodeName Worksheet

Alphabetic | Categorized

(Name)	CodeName
DisplayPageBreaks	False
DisplayRightToLeft	False
EnableAutoFilter	False
EnableCalculation	True
EnableFormatConditionsCalcul	True
EnableOutlining	False
EnablePivotTable	False
EnableSelection	0 - xlNoRestrictions
Name	Sheet1
ScrollArea	
StandardWidth	8.11
Visible	-1 - xlSheetVisible

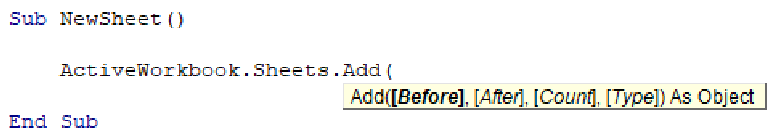


```
Sub ActivateSheet()

    CodeName.Activate

End Sub
```

3.4.2 Edit Worksheet

- Through the use of VBA you can create, copy, delete, clear sheets and more.

  - **Create Sheet**
    - To create new sheet use below syntax. By adding brackets after the syntax you can use optional parameters.



```
Sub NewSheet()

    ActiveWorkbook.Sheets.Add(
                              Add([Before], [After], [Count], [Type]) As Object
End Sub
```

    - Using the syntax above, without optional parameters, will create 1 sheet by default, to create more than 1 sheet at once use parameter count.

```
Sub NewSheet()

    Dim wb As Workbook
    Set wb = ActiveWorkbook
    Dim ws As Worksheet

    'create worksheet and place it before sheet 1
    Set ws = wb.Worksheets.Add(Before:=wb.Worksheets(1), Type:=xlWorksheet)

    'create worksheet and place it after last sheet
    Set ws = wb.Worksheets.Add(After:=wb.Worksheets(wb.Worksheets.Count), Type:=xlWorksheet)

    'create 4 worksheets
    Set ws = wb.Worksheets.Add(Count:=4, Type:=xlWorksheet)

End Sub
```

    - To create sheet with custom name, add name into the syntax.

```
Worksheets.Add().Name = "MySheet"
```

- **Copy Sheet**
  - Similar to create Copy sheet syntax has optional parameters.
  - **Before**: It's an Optional parameter. The worksheet will be Copied to before the specified worksheet. Then we can't specify after parameter. **After**: It's an Optional parameter. The worksheet will be Copied to after the specified worksheet. Then we can't specify after parameter.

Note: *If you **don't specify** either **before** or **after**, Excel will create **new workbook** that contains the Copied worksheet. This ability is useful if you wont to create resulting workbook.*

```
Sub copysheet()

    'copy worksheet named Sheet1 and place it before sheet number 1
    ActiveWorkbook.Worksheets("Sheet1").Copy Before:=Worksheets(1)

    'copy worksheet named Sheet1 and place it after sheet number 1
    ActiveWorkbook.Worksheets("Sheet1").Copy After:=Worksheets(1)

    'copy worksheet into new workbook
    ActiveWorkbook.Worksheets("Sheet1").Copy


End Sub
```

  - You can copy sheet into specified workbook.

```
Sub copysheet()

    Sheets("Sheet1").Copy Before:=Workbooks("YourWorkbookName.xls").Sheets("Sheet3")

End Sub
```

- **Delete Sheet**
  - To delete sheet, you can use below syntax.

```
Sub Delete()

    Sheets("Sheet2").Delete

End Sub
```

- **Clear sheet**

- To clear content of a sheet you can use ClearContents command, but be aware that unless you specify this command, Macro will clear all 1,048,576rows and 16.384 columns. As such it is advised to highlight which rows you want to clear (such as below example) to avoid slowing down your macro.
- **To clear only content (no formats) use;**

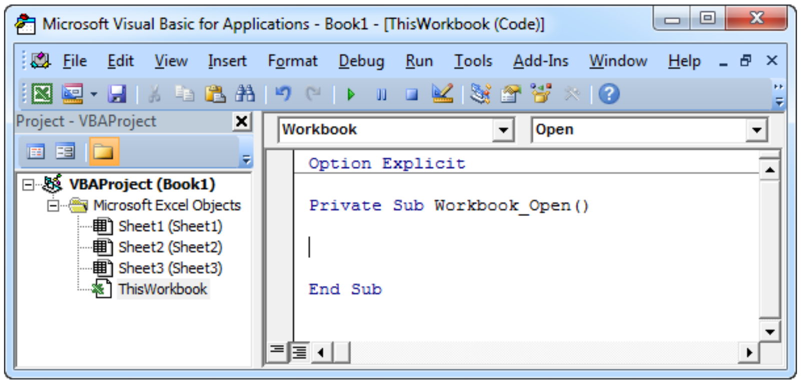
```
Sub Clear()

    Sheets("Sheet1").UsedRange.ClearContents

End Sub
```

- **To clear all, use;**

```
Sub Clear()

    Sheets("Sheet1").UsedRange.Clear

End Sub
```

3.4.3Events

- There are multiple event procedures, below you'll see the most popular ones.
  - **Workbook Open Event**
    - Code added to the Workbook Open Event will be executed by Excel VBA when you open the workbook.
    - In VBA editor click on ThisWorkbook choose from drop down menu Workbook and Open.


Microsoft Visual Basic for Applications - Book1 - [ThisWorkbook (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Project - VBAProject
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook
Workbook
Open

```
Option Explicit

Private Sub Workbook_Open()

End Sub
```

Note: *that as with all the code in VBA this syntax can be selected in above shown drop down menu (Workbook, Open), not just written in.*

- Add the following code line to the Workbook Open Even;

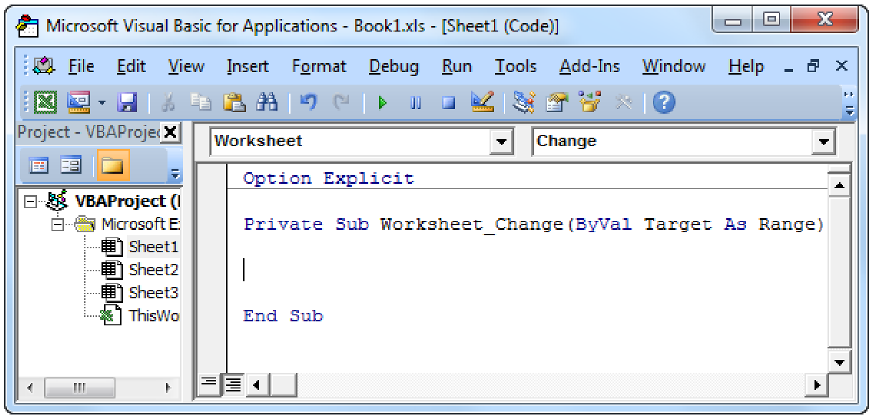
```
MsgBox "Good Morning"
```

- If you were to save and reopen excel, you'd see a pop up message.

- **Workbook End Event**
  - Similar logic as with Open event applies to End event.

- End and Open events can be used to ensure that excel saves itself, creates copy before closing, refreshes certain data, clears sheets, etc.

- **Worksheet Change Event**
  - Code added to the Worksheet Change Event will be executed by Excel VBA when you change a cell on a worksheet.
  - Select Sheet1 in VBA editor
  - Choose Worksheet from the left drop-down list. Choose Change from the right drop-down list.


Microsoft Visual Basic for Applications - Book1.xls - [Sheet1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Project - VBAProje
VBAProject (
Microsoft E
Sheet1
Sheet2
Sheet3
ThisWo
Worksheet
Change

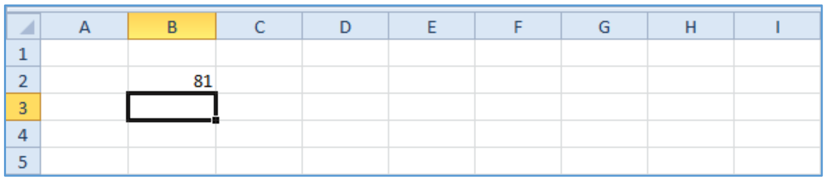
```
Option Explicit

Private Sub Worksheet_Change(ByVal Target As Range)

End Sub
```

- The Worksheet Change Event listens to all changes on Sheet1. We only want Excel VBA to do something if something changes in cell B2. To achieve this, add the following code lines;

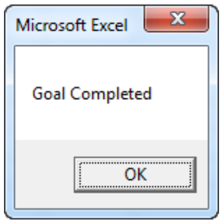
```
If Target.Address = "$B$2" Then

End If
```

- We only want Excel VBA to show a MsgBox if the user enters a value greater than 80. To achieve this, add the following code line between If and End If.

```
If Target.Address = "$B$2" Then
    If Target.Value > 80 Then MsgBox "The bumber is giher than 80, please correct"
End If
```



	A	B	C	D	E	F	G	H	I
1									
2		81							
3									
4									
5									



Goal Completed

- As you can see Change event can be very useful if you have data that requires constant check such as in trackers.

4.0 Closing Exercise

Create Macro as per below requirements

1) Create Button that would start the macro
2) Pop up message will ask you how many rows are in the source
3) Correct the formatting
    - Make sure that each word in name has first letter capital
    - All letters in State have to be capital
    - Format start date
4) Split Full name to get family name into a new separate column named “Last Name” and make sure that the Last name is all capital letters
5) Get the name of month out of the Start date and place in separate column named Month
6) Get end date by adding 3 months to start date into separate column named “End date”
7) Round the Salary to 2 decimals
8) Calculate base pay by “salary – (bonus1 + bonus 2)” into column called over pay
9) Create an Event where if you close the excel it will clear the data from sheet1
10) Create an event which will tell you “Good bye” when closing
11) Add pop up message that will confirm that the macro finished

5.0 ADDITIONAL KNOWLEDGE

Functions - https://www.excelfunctions.net/vba-functions.html
Worksheets - https://trumpexcel.com/vba-worksheets/