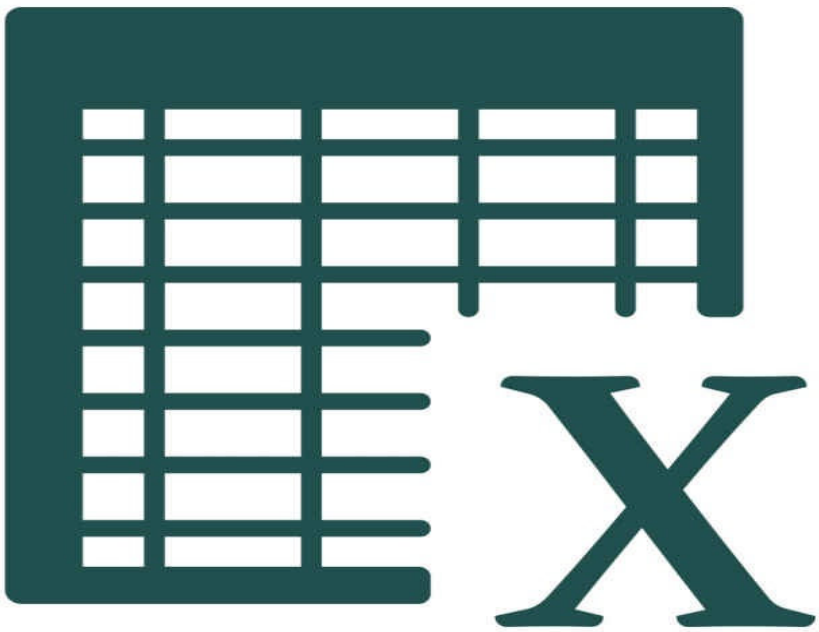
EXCEL VBA

Step-By-Step Guide To Learning Excel Programming Language For Beginners



JASON JAY

EXCEL VBA

Step-By-Step Guide To Learning Excel Programming Language For Beginners

Jason Jay

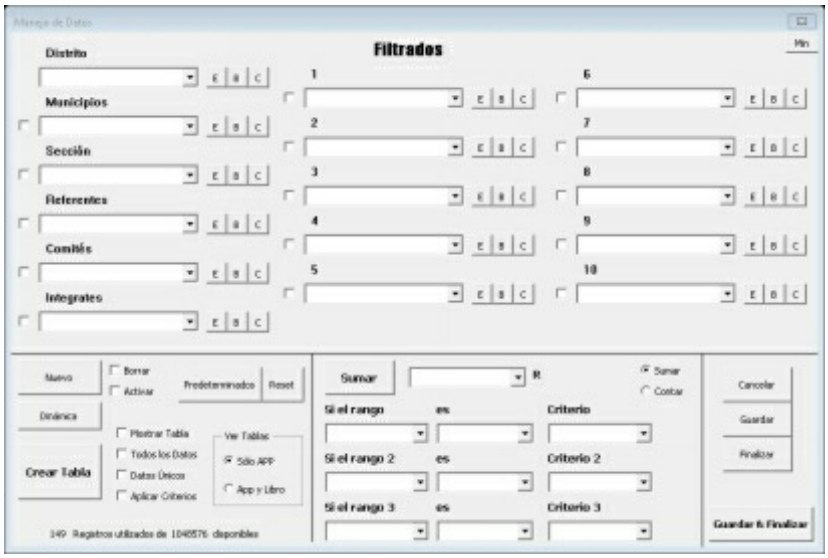
TABLE OF CONTENT

Introduction

If you already know how to use Microsoft Excel but there're a few things you can't do, it is time to learn the strongest functionality it has, Visual Basic for Applications (VBA).

Visual Basic for Applications is a programming language incorporated in Microsoft Excel, Access, PowerPoint and even Word, which let you do all things you already know about them and much more. For example, you want that every time you open a specific Microsoft Word file it writes automatically the current date two lines below where you left last time. Or maybe you want a whole spreadsheet of Excel without formulas on it and still applying them as if they were there. How would you do that? All these things and much more are done with Visual Basic for Applications for Microsoft Office.

Look at the example below:



It looks like a program made for analysis, and it does. Guess what program it is? Probably you are thinking it is not any Microsoft Office

program, but let me tell you that it was made with Microsoft Excel, how would you do something like that without programming? There's no way!

You'll learn much more than that and will be able to create your own programs using Visual Basic for Applications (VBA).

If you need a very specific program for your business analysis, something for personal use, or even just for having fun, you need Visual Basic for Applications now!

CHAPTER 1
VBA Developer TAB

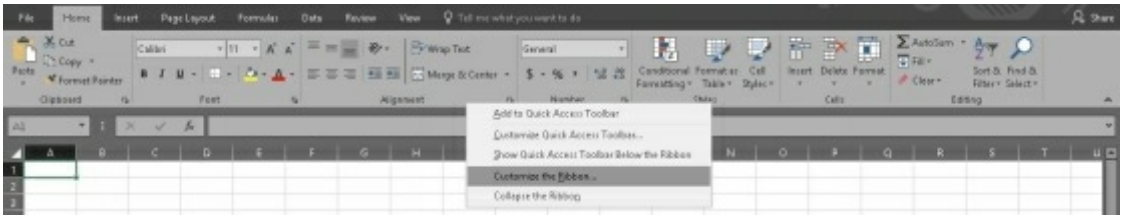
Every programming language has similarities between their fundamentals. The functions IF, Then, Loop, Close, Open, are just some of them. We'll learn the basic ones first; it will be necessary to understand how VBA works.

We'll use Microsoft Excel 365 for this instructions and examples, however, from Microsoft Excel 2007 onwards it will work the same.

Accesing to the Developer TAB

Microsoft Excel doesn't show the Developer TAB by default. It only has File, Home, Insert, etc. But there's no one called Developer. To access to this TAB there are different options, but we'll show the easiest one.

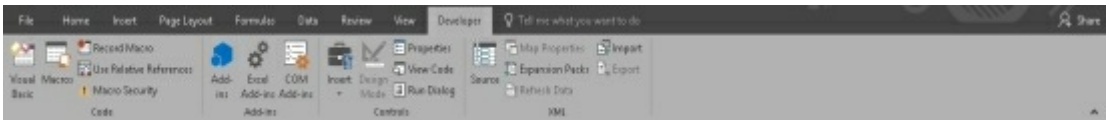
1. Right click to the Ribbon (any part inside the red box, except the buttons).



2. Select Customize the Ribbon

3. Enable the Developer checkbox and click the OK button.

4. You should see the Developer TAB available now.



Quiz 1

1. **How you Access the Developer TAB?**

   a) It is available by default in Excel.

   b) Right click on the Ribbon, Customize the Ribbon, enable the Checkbox for Developer and Accept.

   c) Go to file, Options, Advanced and Enable the Developer TAB.

CHAPTER 2
Macros

Creating a Macro

You'll see a few options available in the Developer TAB, by now we'll start to use the *Record Macro* button.

A Macro is an automated sequence which will apply every time you play it. Let's see a practical example of it:

Imagine that in your job you do the same process every morning. It takes some valuable time and even you're getting bored of that.

The process is the following:

a) You receive a Microsoft Excel file from your boss with some data and you need to write the date using Year, Month and Day in different columns.

You do this because it is the format your job needs and you've been adding the same values every day for a few years.

In this case an semi-automated process would be helpful. Excel gives that option to all of us with Macros. A Macro is a semi-automated process which let you run a specific task using a shortcut.

To create a Macro, follow the sequence below:

a) Click the Record a Macro Button.

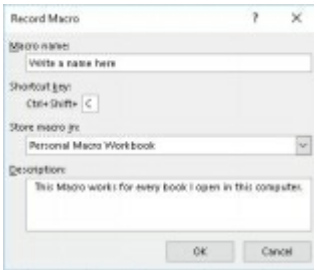
b) Write a name for your Macro. (Needed)

c) A shortkey which every time you press will Run the Macro. Be careful, don't add Ctrl + C or Ctrl + v, otherwise it won't copy or paste anymore, but run the Macro. In case you want a more specific shortcut, hold the shift key as you press a letter. For example, ctrl + shift + c. To make it work, don't press ctrl as you add a short cut. (Optional)

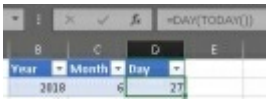
d) Store Macro in: Personal workbook: Will be available for all the files you open with Excel on that computer; New Workbook will be available for a new file only. This workbook, will apply only to the current open file. (Needed to choose one)

e) Write a description about what that Macro does. (Optional)

f) Click Ok.



g) Start doing everything you always do, which would be adding the current date in this case.



h) Once you finish, go back to the Record Macro Button, which now is called Stop Recording. Press it and now should be saved.

This would be a very simple Macro, it only adds the current date, but what would you think if you also need to import data from a web page which is updated every hour, and need to classify it using a few charts, and you do the same process several times a day. No doubt, a good Macro would be useful.

The process to get it any Macro, is the same we've followed. There's only an important thing to consider when creating one, it is to choose between using *Relative References* or not.

Relative References

The Relative References button is just below the Record a Macro Button. Once you click on it, it remains active until you click on it again. It is used to record macros in which the process should be applied to different ranges instead of one already set.

Its functionality is very useful. A macro recorded without relative references will always repeat the process on the same cells used when recorded. But if you use relative references, the macro will run from the active cell. Using the example above, what If you need the dates written on cells F4:H4 instead of B2:D2? The only thing you should do is to select F4 and run the Macro. Or select any cell you need and run it. But you need to record the Macro using Relative References, and then select the cell and run it, otherwise it wouldn't work.

Running the Macro

There are a bunch of ways to run a Macro. Let's see the first: Run this one by clicking on the Macros Button, then click on it to run it.

Maybe it is not as practical as we expected, however, we'll add more functionality and make it easier to run in the next step.

Saving a Macro-Enabled Workbook

Once you have added some Macros to your worksheet and try to save it you'll get a notification like the following:

This might be a little tricky, because most people would attempt saving the file without reading this notification:

The following features cannot be saved in macro-free workbooks:

°*VBA project*

To save a file with these features, click No, and then choose a macro-enabled file in the File Type list.

To continue saving as a macro-free workbook, click Yes.

Most people would just click Yes, and according to this message they wouldn't save the file with their Macros, but as Macro free. It would make you lose all your Macro work.

To save the Macro, just click *No* to the message above, then select save as Excel Macro-Enabled Workbook.

Click on Save, and it is done!

Once you open it again, you should see a message saying *Macros have been disabled,* and a Button saying Enable Macros. Click on it and won't have further problems. If you don't click on it, you won't be able to work with VBA, at least you enable them on Macro Security in the Developer TAB, or follow the steps on Chapter 5: Macro Security.

Quiz 2

1. **What is a Macro?**
    a) It is an Excel Formula
    b) It is a shortcut which runs a recorded process.
    c) It is a built-in process included in Excel.

2. **How to create a Macro?**
    a) Clicking on the Visual Basic Button
    b) Clicking on the Macros Button
    c) Clicking on the Record Macro Button

3. **What is Relative References for?**
    a) It is to Record a Macro without set specific cells.
    b) It is to Record a Macro with specific cells.
    c) Without it a Macro isn't editable.

4. **How to Run a Macro?**
    a) Clicking on the Macro Button
    b) Clicking on the Record Macro Button
    c) Clicking on the Relative References Button

5. **How to save a workbook with Macros?**
    a) Just save the file normally, the Macros will be saved.
    b) You'll get a notification, in which we should be denied and then

select save as Macro-Enabled Workbook.

c) You'll get a notification, in which we will be notified that we are saving a Macro-enabled workbook, then just accept to save.

CHAPTER 3
Starting with VBA

Macros are fundamental to be introduced to VBA.

Let’s see why by following the process:

1.-Create a new Macro without relative references.

2.-In the process select the cell A1, write a number 1, and press enter.

3.-Stop Recording.

Next to the *Record Macro button*, there’s another one called *Visual Basic*. Click on it and you’ll see a code like this:

```
    Range("A1").Select
        ActiveCell.FormulaR1C1 = "1"
Range("A2").Select
```

Congrats, you have some Visual Basic Code now. It means a few orders:

Select Cell A1

Write the number 1

Select Cell A2

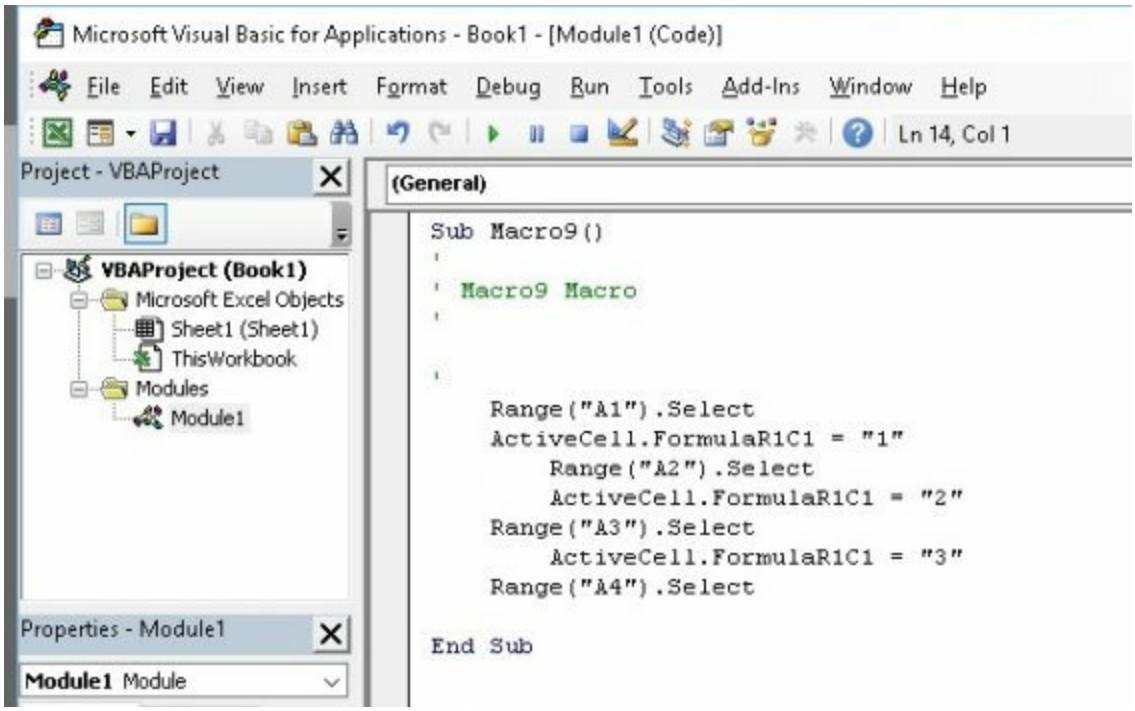
It is exactly what we did. But now, we’ll edit the code so that the Macro does

something else.

If we see, there’s a pattern, which is: select, write, select. So, we could continue the pattern by adding a few more things directly, like this:

```
    Range("A1").Select
        ActiveCell.FormulaR1C1 = "1"
Range("A2").Select
        ActiveCell.FormulaR1C1 = "2"
    Range("A3").Select
        ActiveCell.FormulaR1C1 = "3"
    Range("A4").Select
```

Now that you added this, run the macro, but now you’ll see a second way to run it. Press the green button above the code. Once you click on it go to the Excel Spreadsheet by clicking on the Excel symbol. You’ll see that the cells A1:A3 are filled with the numbers we wrote and the Cell A4 is selected.



It may look very complex, and it is like that because humans use to do things that machines wouldn't. I needed to select cell A1, A2, A3 and son on. But does Excel really need to select it to write a simple number?

Let's try it by adding the following code:

```
Range("A1") = 1
Range("A2") = 2
Range("A3") = 3
```

That's it! Excel can skip steps humans can't. So, it makes it much more faster and easy to accomplish its work.

What if I need to fill the cells up to 100?

Now, I think that if I need to follow this pattern until A100 is going to be hard. What should I do?

Let's add this:

```
Range("A1:A100") = 1
```

Oops! It adds a number 1 to all cells from A1 to A100. Let's try another thing:

```
Range("A1:A100") = 1 + 1
```

It adds a number two instead. So, How do I tell Excel that I want it to fill cells in sequence?

There are several ways. One is to record a Macro and during the process add a number 1 to the cell A1, then hold right click on the small square and scroll down until you select A100. Select fill series. Go to Visual Basic and you'll see a code like this:

```
    Range("A1").Select
        ActiveCell.FormulaR1C1 = "1"
        Selection.AutoFill Destination:=Range("A1:A100"),
Type:=xlFillSeries
    Range("A1:A100").Select
```

It works! I think we already have a great idea about how to get the VBA code we want. Record a Macro, and there it is. However, there're a lot of functions that Macros can't offer. For example, fill up cells in sequence according to the number written in cell B2, and once it changes fill up the sequence again according to that number. So, if there's a number 1 it will be filled one by one, if I change it to five, it'll go five by five and so on. How are you going to record that in a Macro? What if it is not just filling up a sequence, but a Financial matter, working with real numbers and you need to solve a problem like this fast?

Recording a Macro isn't enough always, but in most cases, it helps.

Visual Basic Application (VBA) is a programming language, but it is not necessary to know VBA code or computer programming if the Macro Recorder does what you want.

You should know that when you record a Macro it records even your mistakes, and it will repeat them when you run it. If you want to solve a problem like this you have two options:

1. Record the Macro again.
2. Edit the VBA code.

   Remember that recording a good Macro or writing a good VBA code, will make Excel to run smoothly. Otherwise, you can expect a not responding message until it finishes or maybe it even could stop working.

We'll focus on things you can't do using only recorded macros. So, you can learn how powerful Visual Basic for Applications is in Microsoft Excel.

Insert Form Button

We have some Macros recorded now. But we see that to run them it is not that handy, so what if we add a button, and every time we press it on the macro runs?

To do that, click on the insert button in the Developer TAB. You'll see it displays two boxes, called *form controls* and *ActiveX Controls*. Let's select the Form Control Button, then wherever you want on the spreadsheet hold and drag to create the size of the button. Once you stop holding you'll see a new window in which you will be able to choose the Macro you want the Button to run. Select it and it's done.

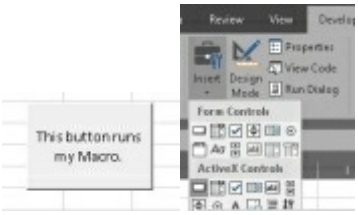
If you want to change the name of the button just click on it twice slowly or right click, edit text. To move the position of the button, just right click, and then hold and drag with the left click.

Easy, don't you think?

Simple things a Macro can't do.

Insert ActiveX Button

Let's add an ActiveX Button. To do this in the Developer TAB click on Insert, and then select ActiveX Button. Hold and drag to create the button in the spreadsheet. This kind of button are very different than Form Buttons, because these ones work directly with VBA.



Let’s add some functionality which can’t be done using Macros only. We want that button to create a sequence of numbers according to the value inserted in cell B2. I mean, if I add number 1 then I’ll see a pattern going one by one. If I write 5, then it will go five by five, and so on, from A1 to A100.

This is a great opportunity to explain Variables.

Variables, Do and Loop.

A variable is an algebraic term, which is usually represented by a letter and its value varies, for example, x + y = 10, in this equation there are two variables, X and Y. They can vary to equal 10. If X = 5 and Y = 5 then x+y=10 , and it is also correct if X= 8 and Y=2, because X+Y= 8+2 = 10. Etc.

In VBA we declare variables too. In this case we’ll add some functionality to the ActiveX Button we added in the previous step.

First, we’ll need to choose the correct numeric variable.

These are the most types of variables, check them out and see their storage size and range by now:

Data type	Storage size	Range

Byte	1 byte	0 to 255
Boolean	2 bytes	**True** or **False**
Integer	2 bytes	-32,768 to 32,767
Long (long integer)	4 bytes	-2,147,483,648 to 2,147,483,647
Single (single-precision floating-point)	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double (double-precision floating-point)	8 bytes	-1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/-0.0000000000000000000000000001
Date	8 bytes	January 1, 100 to December 31, 9999
Object	4 bytes	Any **Object** reference

String (variable-length)	10 bytes + string length	0 to approximately 2 billion
String (fixed-length)	Length of string	1 to approximately 65,400
Variant (with numbers)	16 bytes	Any numeric value up to the range of a **Double**
Variant (with characters)	22 bytes + string length	Same range as for variable-length **String**
User-defined (using **Type**)	Number required by elements	The range of each element is the same as the range of its data type.

In this table, we see that each variable has different ranges, some larger than others, and in the same time the size storage varies too. The smallest unit of memory available is called a byte, and according to this table it goes from number 0 to 255.

Let’s do this to start:

1. Enable the Design Mode Button, which is next to the Insert Button in the Developer TAB.
2. Double click on the ActiveX Button.
3. Now you’re ready to write the code for it.

Private Sub CommandButton1_Click()

End Sub

That’s what you should see.

Now let’s write some code between those lines we see:

```
Private Sub CommandButton1_Click()
        Dim X As Byte
        Dim Y As Byte

            X = 1
            Y = Range("B1")
    Do
                Range("A" & X) = Y
             X = X + 1
             Y = Y + Range("B1")
    Loop Until X = 101
End Sub
```

This is the meaning of this code. It says:

Private Sub CommandButton1_Click()

Private means that you won’t find the code in the Macros Button. CommandButton1 is the default name given to the ActiveX Button, and Click means that the code below will be applied only when you click on it.

Dim X As Byte

Dim declares variables, So, it says that X is a variable that will be used with numbers from 0 to 255, because "*Byte*" only accepts that range according to the table above.

X = 1

I added the initial value for the variable, which will be 1.

Y=Range("B1")

Means that the value of the variable Y is the same value as cell B1.

Do

Means do this.

Range("A" & X) = Y

We already declared X as a variable with a value of 1. So, the Range AX means Range("A1")=Y, and Y is the Range ("B1"). If I add the number 5 to B1, then Y = 5. So, finally it is Range("A1")=5.

X=X +1

It means that X=1+1 because in that moment X=1, so finally X=2. The next time this process repeats, it will be X=X+1 again, but this time X=2, so finally X=3, because X=2+1, and so on.

Y = Y + Range("B1")

This will follow the same process as the last step, but with the value added in Range("B1") which will be that one we added. If B1 has a 5, then it will add another 5, following a sequence according to what we wrote on B1.

Loop Until X = 101

It means that the same process between Do and LOOP will be repeated until X=101.

End Sub

Means the end of the procedure.

Let's try it now!

Add the number 1 to the cell B1 and press the button. You'll see that it fills up the cells from A1 to A100 one by one. Try it with two, and you'll see that it adds from A1:A100 two by two. Finally add the number 3 and press the button.

Congrats! You've found your first bug! It is an overflow.

Bugs are errors when one writes a wrong code. In our example the code works fine with numbers from 0 to 255 only, and once we ask the file to go 3 by 3 one hundred times we are expecting it to go from 3 to 300, which is more than 255, so it causes an overflow.

To fix it, we have two options, we increase the value of the variable or we let Excel to add it automatically.

Let's consider the possibility of increasing the capacity of the variable, we can change it from byte (0 to 255) to integer (-32,768 to 32,767). It will be great if we were pretty sure that we won't work with higher numbers of

32,767, or even change it to “long” which goes from -2,147,483,648 to 2,147,483,647. It would be great to work even with millions, but let’s imagine that someone needs to put billions in B1. In that case even a long variable wouldn’t be enough. So, the best way to solve this is don’t declare the variable and leaving the code in the following way:

```
Private Sub CommandButton1_Click()
    Dim X As Byte
     X = 1
       Y = Range("B1")
   Do
    Range("A" & X) = Y
       X = X + 1
     Y = Y + Range("B1")
   Loop Until X = 101
End Sub
```

Or declaring the value as Variant, which let any number or letters.

```
Private Sub CommandButton1_Click()
    Dim X As Byte
    Dim Y As Variant
     X = 1
     Y = Range("B1")
```

```
    Do
      Range("A" & X) = Y
      X = X + 1
      Y = Y + Range("B1")
    Loop Until X = 101

End Sub
```

What is the advantage of declaring variables as byte, integer or any other?

Maybe you are wondering why to declare a variable as byte or integer, etc. If Excel can add one automatically as in the example we've seen?

The answer is simple and important to know. Let's remember that a variable as byte stores in RAM Memory only 1 byte, which is the smallest amount of memory possible in Excel, but a variant declared as variant stores at least 22 bytes. If we don't declare a value in which we are sure won't add more than the needed values, it will consume more RAM, it will cause a program to run slow in very simple tasks.

For example, I need to declare 1000 variables as byte. It would consume 1 Kilobyte of RAM only.

But By default, if you don't supply a data type, the variable is declared as *Variant* data type, which consumes 22 bytes + string length, string length is for letters.

So, 1000 variables not declared would be automatically declared as variant, and it would consume 22 kilobytes. It is 21 empty kilobytes, which is 95% of

not available space and don’t used at all in case we would have required them as bytes instead. If you are going to program in VBA and want your program to run smoothly, you better learn what kind of variable to declare. Otherwise, it will probably take long to load and then will run slow.

Require Variable Declaration Importance

Probably, one of the biggest problems about not declaring a Variable are creating bugs, because we simply type something wrong. Let’s make a simple example of it.

1. Open Visual Basic
2. Double Click on *ThisWorkbook*
3. Add the following code:

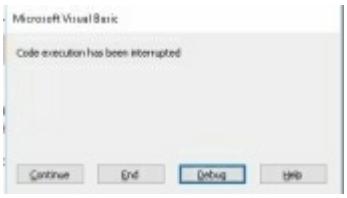
```
Public Sub Infinite()
        myvariable = 200
            myrange = 1
        Do Until myvariabe = 300
            Range("A" & myrange) = myvariable
        myvariable = myvariable + 1
        myrange = myrange + 1
Loop
```

```
End Sub
```

As you see it is a very similar code to one we did before. It should stop once myvariable equals 300, but it continues. Can you see the problem?

If not, run the code. You'll notice that it will go beyond 300, and practically will have no end, or at least until it has no more files. In Excel 365 the total files available is 1048576, so it will go until A1048576 and will enter on an error.

If you don't want to wait until that, Press ESC Key which will interrupt the code execution and then press End Button.



Well, the problem is that we miss just one letter of a variable. We named it myvariable but when coding we miss the letter L : Do Until myvariabe. Excel identified myvariable and myvariabe as two different variables, and automatically assigns them as Variant type.

What would have happened if your code has a lot of code? To find the problem you should look at all your code to find the letter you missed! As you see it would be a big trouble! That's why Microsoft Excel has an option called Require Variable Declaration.

It won’t let you write anything which is not code or a declared variable. To enable this feature do the following:

1. Visual Basic
2. Tools Tab
3. Options
4. Require Variable Declaration
5. Ok.

It will add something above the code, which says Option Explicit. But, you won’t see that change until you start a new project, insert a module or you write it by yourself. That’s why it is important to start a new project always with this option enabled.

In this case, add Option Explicit manually above the code.

```
Option Explicit
    Public Sub Infinite()
            myvariable = 200
            myrange = 1
        Do Until myvariabe = 300
            Range("A" & myrange) = myvariable
            myvariable = myvariable + 1
            myrange = myrange + 1
```

```
    Loop
End Sub
```

After you added this code Run it!

You'll see that it automatically displays a message saying *Variable Not Defined* and even highlight the problem. So, you can see that it will avoid further problems for misspell variables.

APPs Performance

Now, imagine that you really need to fill one by one all the cells from A1 to A1048576, and that it is one of the functionalities your APP should do. Almost always Developers are concerned about creating Apps which run fast. In Excel it is not an exception. Try running the following code and see how long it takes to fill all column A with numbers:

```
Public Sub FillColumnA()
    Dim X As Long
    X = 1
Do Until Range("A1048576") <> Empty
    Range("A" & X) = X
    X = X + 1
Loop
    MsgBox "Finished"
```

```
End Sub
```

When it finishes it will display a message saying “Finished”. It will probably take about 5 minutes, maybe less or more depending on your computer performance. You probably will notice Excel saying that it is not responding, but in most cases it is still working. I don’t want to wait too long!

A good way VBA Developers make Apps run faster, is by disabling the ScreenUpdating. Every change Excel does we are supposed to see it. But, if we don’t want to see it, it will definitively improve its performance.

Let’s change the code to this:

```
Public Sub FillColumnA()
    Dim X As Long
    On Error GoTo A
    Application.ScreenUpdating = False
        X = 1
        Do Until Range("A1048576") <> Empty
            Range("A" & X) = X
        X = X + 1
        Loop
A:
        MsgBox "Finish"
End Sub
```

Run it and time how long it takes, then compare how faster this way is. Don’t

forget to erase Column A before you start. You should notice a big difference! Screenupdating is a great tool when need to increase an App performance. However, now go to the spreadsheet and try to do something. You should notice that it is not possible to do anything, or at least you won't see that, because screenudating = false turned off the visibility changes. So, when you use Application.screenupdating = false NEVER forget to add Application.ScreenUpdating = True at then end of your code, and even better as a backup in case of error do as the code below:

```
Public Sub FillColumnA()
    Dim X As Long
    On Error GoTo A
    Application.ScreenUpdating = False
        X = 1
        Do Until Range("A1048576") <> Empty
            Range("A" & X) = X
        X = X + 1
        Loop
   A:
    Application.ScreenUpdating = True
        MsgBox "Finished"
End Sub
```

MSGBOX

As you saw in the code above, it displays a message saying “Finished”. We can do much more than that!

One of those messages can execute code too! Look at the table below:

Constant	Value
vbOK	1
vbCancel	2
vbAbort	3
vbRetry	4
vbIgnore	5
vbYes	6
vbNo	7

Those are the kind of messages you can display, and once you click on it, it will return the value according to the table above.

Run the following code:

```
Public Sub fff()
    Dim X As Byte
        X = MsgBox("This is the body",
vbYesNoCancel, "This is the title")
        MsgBox X
End Sub
```

It will display a msgbox with a Yes, No and Cancel button. Once you click on any button, X will take that value and will display it on the next MsgBox. So, if you click on Yes, you'll see a number 6, and if you click on No, you'll see a number 7.

This is a great advantage, because we can ask if user want to proceed or not, etc.

If and Select Case

We see that once we are adding a msgbox, we can also create a warning message, or information, or critical, etc. Let's combine one in the code below, so we can figure out how to do it:

```
Public Sub myMessageBox()
Dim X As Byte

X = MsgBox("I'll tell you the button you pressed on", vbYesNoCancel + vbExclamation, "What Button would you press?")
   If X = 2 Then
      MsgBox "You Pressed Button Cancel"
   ElseIf X = 6 Then
      MsgBox "You pressed Button Yes"
   Else
      MsgBox "You pressed Button No"
```

```
    End If

End Sub
```

It will tell you the button you pressed. It means that if you can know what button is pressed on then you can control an action according to it. For Example execute a Macro.

If is the conditional. Like:

```
Public Sub Evaluation()
Dim X As Byte

X = Range("A1")
If X = 5 Then
MsgBox "There's a number " & X
    ElseIf X = 6 Then
        MsgBox "There's a number " & X
    ElseIf X = 7 Then
        MsgBox "There's a number " & X
    ElseIf X = 8 Then
        MsgBox "There's a number " & X
    Else
        MsgBox "There's another value"
    End If
```

```
End Sub
```

So “if” makes a conditional, “elseif” is checked in case the first condition doesn’t meet, and will check as many “elseif” as you add until one condition meet. “Else” will run in case any condition doesn’t meet. So, if we don’t add any number from 5 to 8, we’ll see a message saying “There’s another value”, but if we put a number 8, then we’ll see a message saying There’s a number 8.

If, elseif and *else* are very similar to another kind of code called *Select Case, Case,* and *Case Else* However, this one is more efficient than *IF* in some cases. Let’s do the same than above but using case instead:

```
Public Sub Cases()
Dim X As Byte
X = Range("A1")
  Select Case Range("A1")
   Case 5 To 8
        MsgBox "There's a number " & X
   Case Else
        MsgBox "There's another value"
   End Select
End Sub
```

This last one occupies much less code. It was easier and better. You can do practically the same but is up to you which one you want to use.

Quiz 3

What is a Variable?

a) It is a value which never changes.

b) It is a value which changes.

c) It is a special number.

What is the storage size of a byte variable and its range?

a) It stores 2 bytes and goes from -32567 to 32567.

b) It stores 1 byte and goes from -256 to -256.

c) It stores 1 byte and goes from 0 to 256.

What is the advantage of declaring variables?

a) It consumes less RAM and makes it run smoothly.

b) It consumes more RAM and makes it run slower.

c) It wouldn't work if variables aren't declared.

Is it obligatory to declare variables?

a) Only if Require Variable Declaration is enabled

b) It is obligatory in all circumstances

c) It is not, Excel declares them automatically according to the value added.

Msgbox returns a value depending on the button pressed:

a) We assign the value to each button

b) It returns a value depending on the button pressed

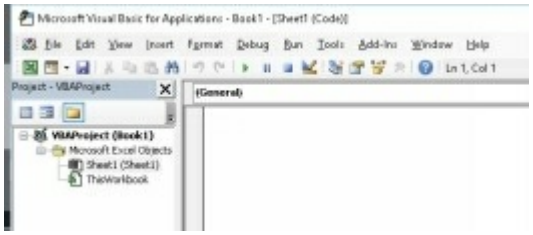
c) Values aren’t returned

CHAPTER 4
Project: Creating a Simple Calculator using ActiveX

We're going to create a simple calculator first. After this project is completed, we'll create a more sophisticated one. To do that, we'll require to know the following:

What is a Module?

When we start writing VBA code, we'll usually start writing on Sheet1. But, in order to understand Modules, Procedures Private and Public, we'll create a visual calculator. First, we´ll do it on a spreadsheet, and then we'll do it as a real program.



A module, is something like a Box, in which we add some code to run when we "call" it. To understand how it works, we'll create our calculator using a few Modules.

How to create a Module

First, add the following ActiveX buttons on a spreadsheet on the table as it is shown on the image. To add the appropriate symbol to each button, it is not

as we did on the Form Buttons. In this case we have to click on the Design More Button, which is in the developer TAB, then right click on each button and you'll see something like the image below, in which we see something called "Name". Most people would think that there's where we can add the correct symbol to see, but that's wrong. Excel identify the Button with that name. Imagine you have several buttons with + as their name. How would either Excel or you identify which one is the correct? So, every button will have a different name, let's add a good one which let us know what it does like In the example I wrote cmdAddition. It lets me know that that it applies to the "+" Button. However, how can I add the sign +?

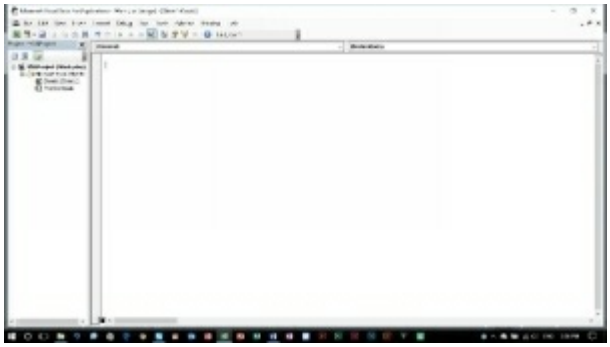
In the image above, you 'll see that there's a property called Caption. That's where You'll add the "+" sign.

You can even play a little bit with the other options, like the Backcolor, Font, Height and even add a picture. I left all options as they were by default.

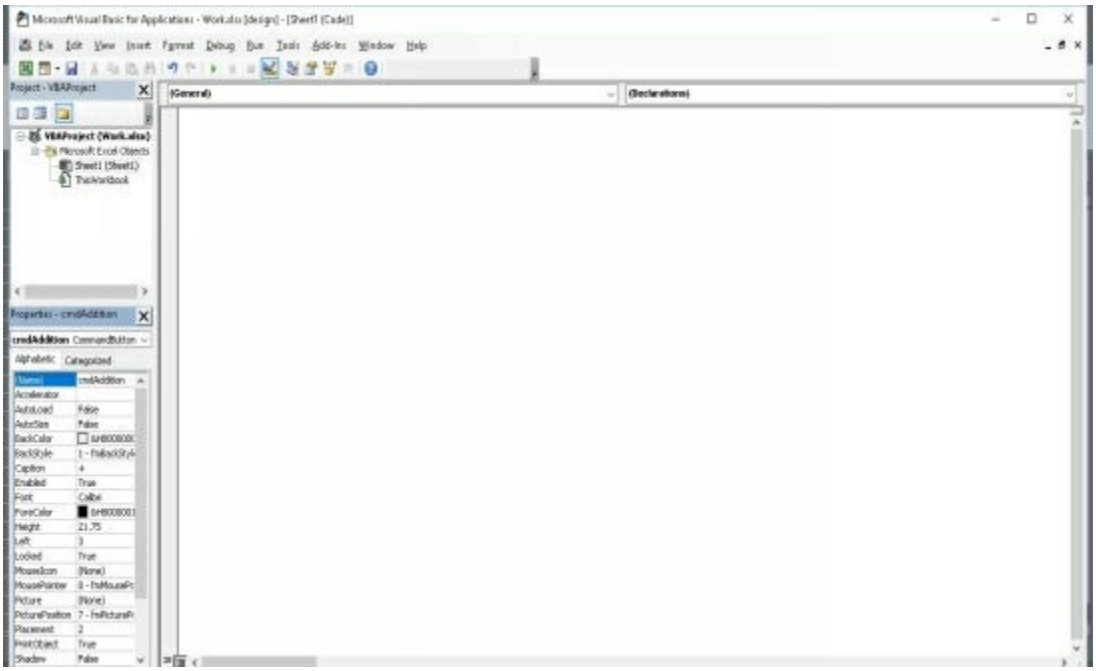
Once you repeated the process for all the buttons, let's start with the next steps. Let's create a Module!

Follow these steps:

1. Open Visual Basic through the Developer TAB. By default you'll see something like this:



2. Click on "View TAB" and select "Properties Window". We'll always use this window a lot. Now you should see something like what's shown inside the red square:



3. Let’s add our first Module. Click on Insert TAB, then Module.

4. You’ll see a New Folder Called “Modules” with a file there Called “Module 1”:

5. Select it and in the Properties Window Change its name to Addition, then repeat the process until you have created a Module for Minus, Division and Times.
6. Double click on the Module “Addition” to open it.
7. You’ll see that it displays an entire blank sheet. We need to add a Procedure! To do that, click on Insert TAB, then Procedure. It will

display a window like this, add a name and leave the default options:

8. Click OK and you'll see this:

9. Between those lines write the following code:

Range ("A2") + Range("B2") = Range("C2")

10. Now run it by clicking the green triangle above.

11. It is displaying an error saying: Invalid use or Property. Maybe it is because we haven't added any number in the cells A2 and B2. Add them and run it again.

It is still having a problem! Can you see what's wrong?

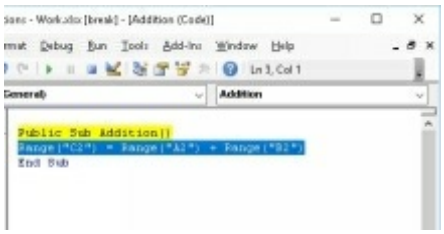
Welcome to your second bug! The problem here is a very usual one. We are asking Excel that cells A2 + B2 are equals to C2 instead of C2 equals to A2 + B2. This problem is just an order problem. You better don't forget this rule!! Always add first the cell you want to be changed, then add the values that you'll need. Like this:

```
Range("C2") = Range("A2") + Range("B2")
```

Run it again with the same green triangle.

OK! It works!

If you see a yellow line which doesn't let you run it, just click stop, correct your code and run it again.



Now you'll see that in the cell C2 we find the result of A2+B2.

Let's complete the other modules repeating the correct process above. Add these codes to do that:

Minus Module:

```
Public Sub Minus()
   Range("C2") = Range("A2")  - Range("B2")
End Sub
```

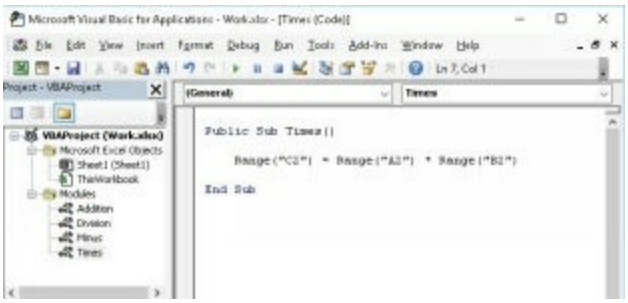
Division Module:

```
Public Sub Division()
        Range("C2") = Range("A2") / Range("B2")
End Sub
```

Times Module:

```
Public Sub Times()
   Range("C2") = Range("A2") * Range("B2")
End Sub
```

You’re supposed to see a file like this:

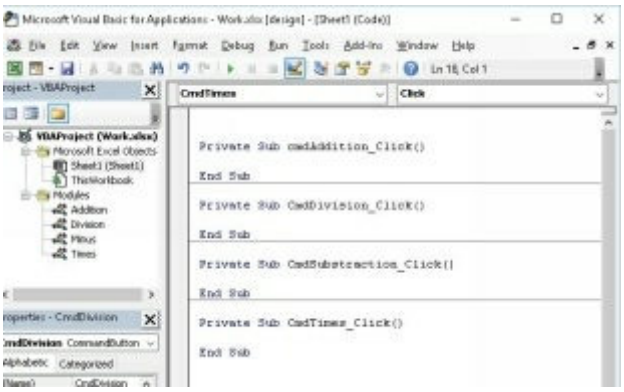


All the modules should work if you run them.

Now, let’s link the Modules to their appropiate button.

To do that just do the following:

1. Go to the spreadsheet
2. Click on Design Mode
3. Double click each button. You’ll see that every time you do it, it adds some code to the Sheet1(Sheet1). Finally, it should look like this:

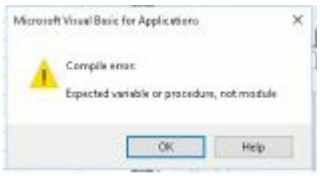


Now, let’s do the process called “Calling”. To do that will just write every Module between its corresponding lines:

```
Private Sub cmdAddition_Click()
        Addition
End Sub

Private Sub CmdDivision_Click()
        Division
End Sub

Private Sub CmdSubstraction_Click()
        Minus
End Sub

Private Sub CmdTimes_Click()
```

```
        Times
    End Sub
```

After you wrote the code above try pushing a button:

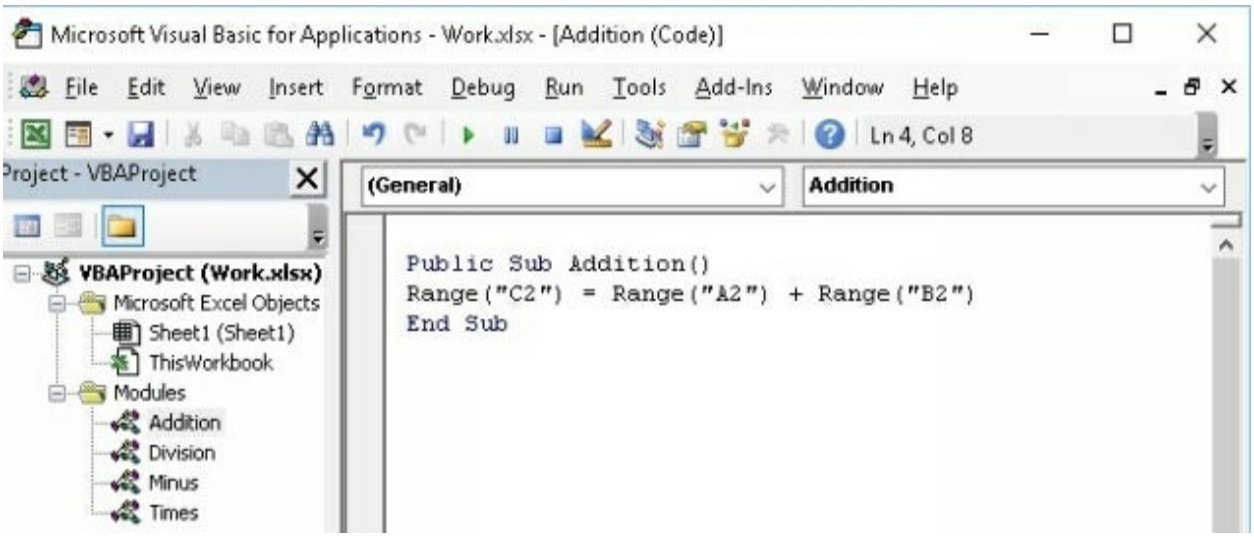


Welcome to your third bug!!

Why is this happening? Let's read the message.

It says that it is not expecting a Module, but a Variable or Procedure.

The problem in this bug is that Modules and Procedures in this example are called the same and that's a big mistake. Nothing should have the same name when programming!



Let’s fix it fast. Just add manually this time a number 1 after each sub procedure in each module.

Public Sub Addition1()

Range("C2") = Range("A2") + Range("B2")

End Sub

You also should change the written code on Sheet1. Just add that one we’ve just added to each sub procedure in the modules:

Private Sub cmdAddition_Click()

Addition1

End Sub

Private Sub CmdDivision_Click()

```
Division1
End Sub

Private Sub CmdSubstraction_Click()
Minus1
End Sub

Private Sub CmdTimes_Click()
Times1
End Sub
```

Try it now by pushing every button and see if it works!

Great!!

We are supposed to have a much better idea about Modules and Procedures. We'll always use them. However, maybe you noticed that some of them are called Public and others are called Private.

What does it mean?

PUBLIC means that that procedure can be called from anywhere. You'll even notice that all of them are even in the Macros list, but those saying PRIVATE aren´t.

Try its meaning by changing just one or two Modules, change the word Public to Private, and try run it.

As it was a Private one, it couldn't find it. Let's change the word Private back to Public, you'll see that it will work.

So, Private ones can't be called, Public ones can be called, even from the Macros list. In other words, Privates can't be linked, but Publics can.

We've seen the most basic functions of VBA. You should have a great idea about VBA now. However, let's do something much more professional: A real calculator.

Adding Letters?

Before we start the other calculator, do this experiment which will be interesting for you to know, and in the same time, very useful. In the last calculator we've made try adding two letters instead of numbers and Add them.

You'll notice that you can add letters!! A + A = AA !!?? It shouldn't work if you make a subtraction, division or multiply.

It happens because using " + " in VBA is not the same than the formula =SUM(). Let's remember that by now, because it will very useful for the next project.

Quiz 4

What is a Private Procedure?

a) It means no one can see the Code because it is hidden.

b) It means it can't be copied and pasted

c) It can't be called from another part of the code

What is a Public Procedure?

a) It means that it can be called and executed from any part of the Workbook

b) It can be seen online

c) It means you can't protect it with passwords.

CHAPTER 5
Project: Calculator using Forms

Review

This time we'll use much more Visual Basic for Applications functionalities. Now we will understand why it is called Visual Basic.

First than anything let's review a few things important to consider:

1. Choose the correct Variables to avoid bugs and make the program run smoothly.
2. Write code in the correct order, otherwise it will produce a bug.
3. Never repeat the name of anything in a program. Everything should be identified with a unique name.
4. Hope you noticed blue letters. They are built-in codes used by Excel VBA. Don't erase them or change them without knowledge, or it will produce errors.
5. TIP: Always add names using at least a Capital, and when you call them write their names using lowercase. You'll notice that using this way you'll avoid typing mistakes, because if you wrote it correctly it will add Capitals automatically, otherwise it will remain the same.
6. Constantly try the new code you've written, so you'll be sure that it is working fine.
7. Add enough comments to let you know what your code does in case you need to review anything in the future. We'll learn how to do that.

FORMS

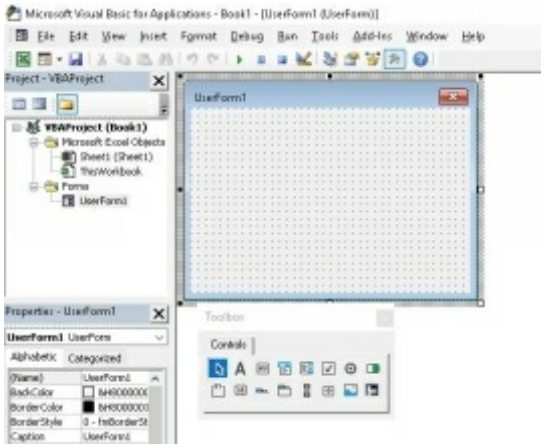
Visual Basic for Applications in Excel has a very attractive feature called

Forms in which we can create a visual application like that we saw in the beginning of this book. That's how we'll create the calculator.

Follow these steps:

1. Open a New Blank Excel Document.
2. Open Visual Basic.
3. Click on Insert TAB.
4. Userform.

You should see something like this:



That's an userform. It will be used to create a Calculator this time. First, you see a Box called "Controls" with several options inside. We'll use it almost all time, so don't close it. In case you do it, ou can open it again by clicking on View TAB, then on Toolbox.

Don´t close the properties Userform either. This will be the main tools we'll use for this project, and In this case, we won't even touch a Excel Spreadsheet.

In the properties window change these values:

Name: CalculatorProject

Caption: Calculator

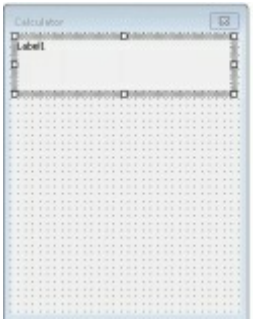
Height: 260

Width: 200

Remember that Caption is the Displayed Title.

In the toolbox select the label, it is an A next to the Arrow.

Once it is selected, click on the upper side of the form, hold and drag to make a rectangle which will be our screen for the calculator.



Now, select the label, and change its name in the properties. Name It *Display*. Then, erase the caption value.

In the ToolBox, select CommandButton, then add a series like the following:

Everytime you add a new button, make sure to add it with the following size:

Height: 30

Width: 36

TIP: Make just one, then copy and paste all the buttons you need. Place them in the correct order, then make sure to add each button with the following info:

Name	Caption
CmdDel CmdCE	Del CE

CmdPercent	%
CmdTimes	*
CmdDivision	/
CmdAdd	+
CmdMinus	-
CmdEquals	=
Cmd1	1
Cmd2	2
Cmd3	3
Cmd4	4
Cmd5	5
Cmd6	6
Cmd7	7
Cmd8	8
Cmd9	9
CmdDot	.
Cmd0	0
Cmd00	00

Great!

Commmand Buttons code

Once we have the design of the calculator and their names correctly we can run it and click on every button. You'll notice that it does nothing, because we haven't told Excel what to do with it. Excel doesn't know if I want that form for a Prank or any other thing. So, let's add code to every button.

1. Double click to the button with number 1.
2. It added automatically this code:

    Private Sub cmd1_Click()

    End Sub
3. Add this code between the lines above:

    Display.Caption = cmd1.Caption
4. Run the form.
5. You should see a number one written on the Display!! But, it doesn't add more than just one number 1!! If it is going to be a calculator, we need to add several ones if needed! What's wrong??

The problem here is that the code says the Caption of the Display (which we left empty), is equals to the caption of cmd1, which is a number 1 and it is exactly what it is doing.

So, to solve this problem we must to tell it to add a number 1 after a number 1, and so on.

Display.Caption = Display.Caption + cmd1.Caption

Maybe you got it now. It is exactly what we did almost in the beginning of this book. Do you remember when we make cells increase their value according to the number written on cell B1?

If you want to check it out to review go to **Variables, Do and Loop**. On page 11.

It means that X = X + 1 or in other words, means that Display.caption will add its own value plus the caption of cmd1. So, it may be 1 = 1 + 1 = 11 It

doesn’t have any sense in mathematics, however, do you remember when we added letters to our last calculator project? According to that calculator A + A = AA, so here 1 + 1 = 11. To try to make it clearer, if you type button one it would be 1 = 1 = 1 then let’s suppose you type number two. It would be 1= 1 +2 = 12, now you type number 7: 12 = 12 + 7 = 127 and finally you type number 9: 127 = 127 +9 = 1279 and so on.

Run it with the new code. Great! It should be working now!

You should have all this code:

```
Private Sub cmd1_Click()
Display.Caption = Display.Caption + cmd1.Caption
End Sub
```

Let’s repeat the process to all the numbers and symbols, by double clicking on every button and once it adds some code automatically, write the correct code line between them, look at the following code so you may have a better idea:

```
    Private Sub Cmd0_Click()
        Display.Caption = Display.Caption + Cmd0.Caption
    End Sub

    Private Sub Cmd00_Click()
        Display.Caption = Display.Caption + Cmd00.Caption
    End Sub
```

```
Private Sub cmd1_Click()
    Display.Caption = Display.Caption + cmd1.Caption
End Sub

Private Sub Cmd2_Click()
    Display.Caption = Display.Caption + Cmd2.Caption
End Sub

Private Sub Cmd3_Click()
    Display.Caption = Display.Caption + Cmd3.Caption
End Sub

Private Sub Cmd4_Click()
    Display.Caption = Display.Caption + Cmd4.Caption
End Sub

Private Sub Cmd5_Click()
    Display.Caption = Display.Caption + Cmd5.Caption
End Sub

Private Sub Cmd6_Click()
    Display.Caption = Display.Caption + Cmd6.Caption
```

```
End Sub

Private Sub Cmd7_Click()
    Display.Caption = Display.Caption + Cmd7.Caption
End Sub

Private Sub Cmd8_Click()
    Display.Caption = Display.Caption + Cmd8.Caption
End Sub

Private Sub Cmd9_Click()
    Display.Caption = Display.Caption + Cmd9.Caption
End Sub

Private Sub CmdAdd_Click()
    Display.Caption = Display.Caption + CmdAdd.Caption
End Sub

Private Sub CmdDivision_Click()
    Display.Caption = Display.Caption + CmdDivision.Caption
End Sub

Private Sub CmdDot_Click()
```

```
        Display.Caption = Display.Caption + CmdDot.Caption
    End Sub

    Private Sub CmdMinus_Click()
        Display.Caption = Display.Caption + CmdMinus.Caption
    End Sub

    Private Sub CmdPercent_Click()
        Display.Caption = Display.Caption + CmdPercent.Caption
    End Sub

    Private Sub CmdTimes_Click()
        Display.Caption = Display.Caption + CmdTimes.Caption
    End Sub
```

All this code should let us see every number and symbol on the display. Button equals, Del and CE are those we don't see written on our display, and will give them a different treatment.

First Button CE:

It should clean or erase all the screen. This one is very easy, this is the code:

```
    Private, Sub CmdCE_Click()
```

```
    Display.Caption = Empty
End Sub
```

It means exactly what is described on the very same code. The display will be left empty.

Second, Button Del:

It should erase one by one, from the last number we have added. How would you do that? Maybe, if you are learning VBA you should already know Excel Formulas. So, try to figure out how to do that.

In case you struggle with that I'll give you a shortcut:

Value (Cells as A1 Below)	Formula applied	Result
12345678	=left(A1,len(A1))	12345678
12345678	=left(A1,len(A1)-1)	1234567
12345678	=left(A1,len(A1)-2)	123456

=left() Displays from the left the value, according to the number of letters we told it to do.

=len() counts the number of letters contained on a cell.

-1 It will subtract one from the total of letters or numbers of len().

The result is displayed above. It is clear, so what does happen if we repeat the process several times applied to the same cell?

Repetitions	Value (Cells	Formula applied	Result

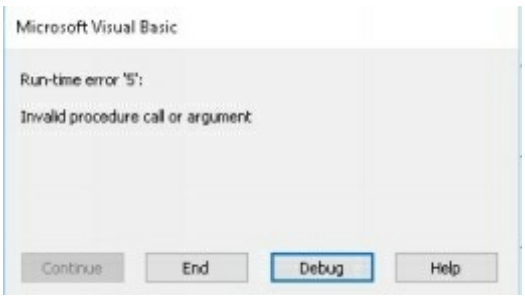
as A1 Below)			
1	12345678	=left(A1,len(A1))	12345678
2	1234567	=left(A1,len(A1))	123456
3	123456	=left(A1,len(A1))	12345

So, It does what a DEL button should do. We need that every time we push on that button it erases only the last letter or number added, as this formula does. Then, we need the button to apply this formula. How to do that in VBA?

First, double click on the button and add the following code:

```
X = Len(Display.Caption) - 1
Y = Display.Caption
Display.Caption = Left(Y, X)
```

That's it!! We just added two variables and used them in a very simple sequence! Try it out. Add a few numbers and then press this button. It should work. However, what does happen if there are no more numbers and you press the button?



That’s obvious that it would produce an error. How to solve this?

That’s easy, add some code including IF:

```
Private Sub CmdDel_Click()
    If Display.Caption <> Empty Then
        X = Len(Display.Caption) - 1
        Y = Display.Caption
        Display.Caption = Left(Y, X)
    End If
End Sub
```

We have added this two lines of code which means that **if display.caption is “not” empty then** apply the code. So, if it is empty wouldn’t do anything.

Excel Formulas on VBA

Third, let’s work now with the equals button:

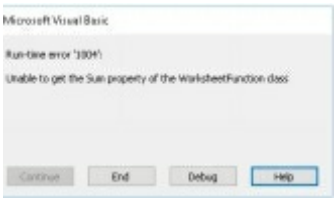
We need it to make all the operation we have added on display.caption. How would you do that?

This one is the hardest of this calculator, and the most important button because it will give the result of everything we’ve done. There’re several ways to do this, but this time let’s use: *Application.WorksheetFunction.* Once you type the dot, you should see a long list. All of those are the

complete list of Formulas which we use in Excel normally. Let's add the following code and see what happen this time:

```
Private Sub CmdEquals_Click()
    Dim X As Variant
        X = Application.WorksheetFunction.Sum(Display.Caption)
        Display.Caption = Empty
        Display.Caption = X
End Sub
```

It should work, don't you think? But it won't.



The first line declares X as a variant which means that could store any value.

X is equals to the SUM(display.caption)

Which should be like X = sum(2+2*4/7)

Combining VBA and a Spreadsheet

If you try that on a cell of a Spreadsheet, it will work. But it is not working using a variable because of several reasons, the main one is because one can't really add a number 12 plus an addition sign, etc. We would need to make a

conversion and some more procedures, but in this case we could just start using the spreadsheet.

We'll use a cell of the spreadsheet, so we could get familiarized combining VBA with Excel Sheets.

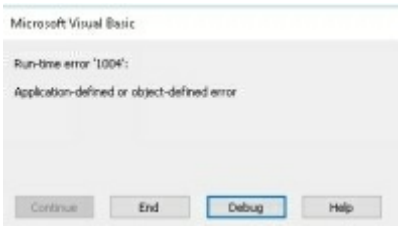
We'll command Excel to convert Display.Caption to a Formula in cell A1.

If you record a Macro and add a Formula, probably the code will be: Activecell.FormulaR1C1 = "="… But we don't need the R1C1 for our code here, that's why we erased it.

```
Private Sub CmdEquals_Click()
    Range("A1").Formula = "=" & Display.Caption
    Display.Caption = Range("A1").Value
End Sub
```

Run the program and try it now! It works!! Now, write a bad code, something like just: 5*

And press =.



Another bug!

That’s why programmers try their apps frequently. You have to do the same always. AS you see we have fixed several bugs only working on a Calculator, but once you get used to code will identify and even prevent them easier. I know that.

To fix this bug, we’ll apply our first Control Errors. So, every time something goes wrong it will be applied instead this kind of windows.

Add the Following code:

```
Private Sub CmdEquals_Click()
    On Error GoTo A
        Range("A1").Formula = "=" & Display.Caption
    Display.Caption = Range("A1").Value
Exit Sub
A:
        Range("A1").Clear
    Display.Caption = "Error"
End Sub
```

It may be easy to understand. On Error Goto A, and we see an A in the middle. So, if something goes wrong the code will be applied from that A to the End, which is End Sub.

It says Erase cell A1 and in Display.Caption show the message Error instead.

We also see above A: something saying Exit Sub. It means that the code will be read until it reaches it only, otherwise the calculator would read all time

what’s for errors too. We don’t want that.

Try it again, write something wrong like 5+ only.

See? It worked!!

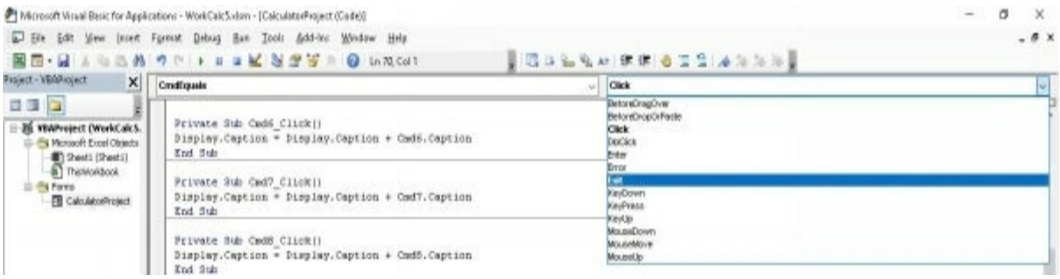
Our Calculator is looking better at every moment.

Starting with Declarations:

Now, I think it is not nice to see the Error message even after if I typed another button:

It would be handy if it erases it automatically once I type any other button. To do that just we’ll add another line code to the equals button. This time it will not be applied when we click it, but when we exit the button which will happen once we type another button. To do that just do the following:

1. Double click to the Equals button.
2. Click to see all the options like in the image and select Exit



Great! Now you've interacted with different attributes each button has! As you see code could be applied if you click on the button, once you hold it down, exit and even if you put the cursor above the button! It is amazing don't you think??

Once you've chosen Exit, you'll see a new code there. Add the following:

```
Private Sub CmdEquals_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Display.Caption = Empty
End Sub
```

Nice! It is working as we wanted now. I think you've seen that coding is just logic, after you learn a few more commands you'll do amazing things.

By the way, You learned how to relate a little bit a spreadsheet and VBA, because we are using the range A1. What if we don't even want to use any cell of a spreadsheet??

As we saw, we can't use variables. Right? Obviously, we can!! We just need to know the way.

Now, you can erase everything from the equals button, change the code from:

```
On Error GoTo A
```

```
            Range("A1").Formula = "=" & Display.Caption
        Display.Caption = Range("A1").Value
Exit Sub
A:
            Range("A1").Clear
        Display.Caption = "Error"
End Sub
```

TO:

```
Private Sub CmdEquals_Click()
        On Error GoTo A
            Display.Caption = Application.Evaluate(Display.Caption)
Exit Sub
A:
            Display.Caption = "Error"
End Sub
```

We didn’t go directly to this solution because by commiting mistakes we get a valuable experience. Maybe, one day you won’t know what function to use and knowing how to link VBA with a Spreadsheet will be very useful to solve your problem.

The code above has this meaning:

Application means Excel.

Evaluate is a method of the application or Excel. It converts a Microsoft Excel name to an object or a value. In other words, in operable values which can be sum, divided, multiplied, etc..

Now, we are not using any spreadsheet at all. So, Why should we leave it open? We are just interested on seeing the App we've created! What can we do?

Open and Close declarations: Displaying a form without looking any spreadsheet

This calculator is working fine. But we don't like to see the whole spreadsheet behind. We don't even want to see Excel opened at all, because It is not a common thing to see when we use an app, so let's take the spreadsheet out!

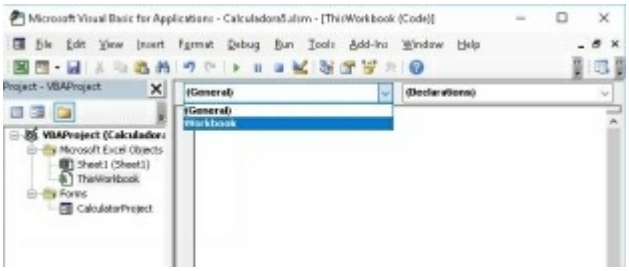
It should look like this:

We just see the Desktop behind, and not any spreadsheet.

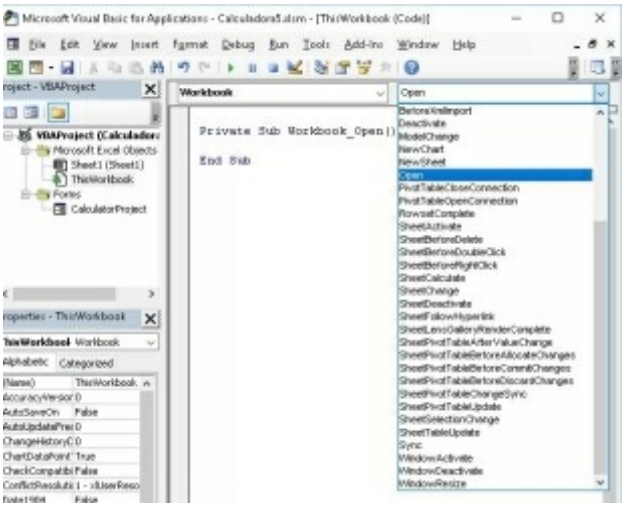
Let’s do it step by step. We’ll learn more about VBA functionalities, and even some Macro securities.

Follow these steps:

1. First than anything, we want that our calculator opens automatically once the file is open. So, in the Folder VBA Project(Calculadora), select *ThisWorkbook.* You should see two options, one says General and the other says Declarations.
2. Open the General one and select Workbook.



3. Now, Declarations Should have be changed to open. Otherwise, select it.



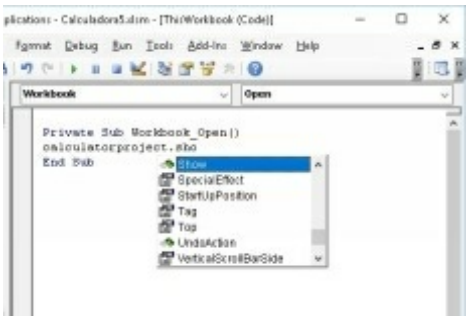
This means that everytime This Workbook Opens, execute the code between:

Private Sub Workbook_Open()

End Sub

4. Now, between those code lines, insert the following:

```
Private Sub Workbook_Open()
    CalculatorProject.Show
End Sub
```



5. Save your file.
6. Close and open to give a try! It should open the Calculator project automatically. That's the Open Event and is very useful from small to big projects.

We still don’t see the app only, but Excel still appears there. So, do you rememeber that in some code above “Application” means Excel? It means that we should write some code saying that Application must be invisible.

Add the following to the code you’ve recently added:

Application.Visible = False

It should look like this:

```
Private Sub Workbook_Open()
    Application.Visible = False
    CalculatorProject.Show
End Sub
```

Save the file, close and open to give a try!

Great!! Now you see the App only!! Once you close your Calculator everything seems to be working fine. However, there’s something still wrong.

Once you close the calculator, Excel seems to close too,however, it remains open but not visible. We closed only the app!! Try to open the same file in which you have your calculator. You’ll notice that it opens without loading anything and doesn’t display the calculator, the reason is that it was open!

We want that once we close the calculator, it also closes Excel. To do this follow these steps:

1. Open VBA.

2. CalculatorProject
3. Double clik on any part of your Form to see the code. Not the caption neither the buttons, but any other part of it.
4. It will open something like this:

   Private Sub UserForm_Initialize()

   End Sub

5. It is very similar to the Open Declaration we learned, but it works with the project only. However, we don't need this by now. We need the closing code for our Form. The way to select it is as we did with *ThisWorkbook.* Go to where it says Initialize, then select the most logic one, which would be *Terminate*.
6. *Once you select it, you'll see a code like this:*

   Private Sub UserForm_Terminate()

   End Sub

7. Remember again that *Application* means Excel? You'll notice once you type Application. (Followed by a dot) that it displays a large list. You can take a look, but be careful

about experimenting. Sometimes, you can make big mistakes by playing with code. Even outside Excel!! Choose *Application.Quit*

Your code should look like this:

```
Private Sub UserForm_Terminate()
    Application.Quit
End Sub
```

8. Save your file and try it out! It should work now! Fast and easy!

Macro Security

Our Calculator seems to work great now. However, try doing the following:

1. 7*7
2. Press Equals Button
3. You see 49
4. Now, I want 49+1. So, type (+) …
5. You'll notice that 49 erases automatically. It is not normal in a calculator, but it should let me add some code after it displayed a result. Let's add the code.

Maybe, we are in a trouble. How are we going to edit the VBA code if

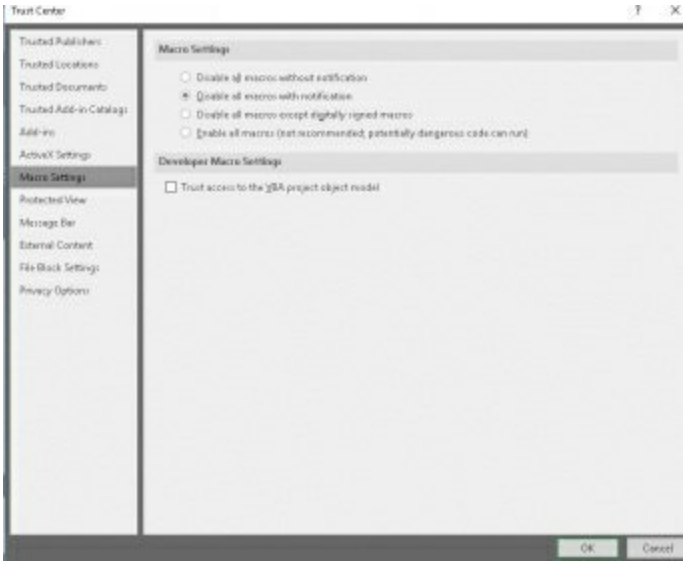
it opens and closes Excel automatically? How are you going to access to that??

This is a great opportunity to understand Macro security.

You should have clicked on the Enable Macros Button to work with VBA as we learned on Chapter 2. Once you click on it, Excel saves a path in which that specific file is allowed to work with Macros. So, to disable the allowed Macros all you have to do is to move the file to another location. Try to do it, and then open it again. You'll notice a message saying that Macros have been disabled.

Another way to do it without changing the location is to change the file name. Try it out and you'll see you receive the same message about Macros.

If you miss it and don't click on Enable Content, or just want to change the Macro Security Options, Go to the Developer TAB and Click on Macro Security. You'll see all the available options for Macros in all workbooks and for that specific file.



Now you can go to VBA and change the code we need.

Comments

Now, we don't want our calculator to erase what is written on our display, unless it is an error message. It will allow us to work more efficiently, so we need to go again to the code which erases that. Do you remember which one does that? Probably you do because they are only a few buttons. However, what would have happened if they were hundreds and hundreds of commands?

It is always convenient to write a few comments which will let us identify easily what things some code does. It is very common to fix a few bugs, add, remove or improve some functionality when programming. So, don't forget to add Comments to let you go faster to the code you need to change.

In this case we need to change the Equals Button. However, this has two codes added. One runs when we click the button and the another once we stop selecting it.

```
    Private Sub CmdEquals_Click()
        On Error GoTo A
            Display.Caption = Application.Evaluate(Display.Caption)
        Exit Sub
    A:
        Display.Caption = "Error"
    End Sub
Private Sub CmdEquals_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Display.Caption = Empty
End Sub
```

The code we need to change is the second one of those above. But first, let's add a few comments to remember what they do for future reference. To add comments all you need to do is write what you want by adding a ‘ in the beginning of each sentence. In the Example below I added two sentences, so I needed to put a ‘ at the beginning of each sentence.

```
Private Sub CmdEquals_Click()
‘This Macro displays an error message in case we type wrong imposible mathematics and click equals
‘otherwise it would run a bug and the program would stop working.
        On Error GoTo A
```

```
            Display.Caption = Application.Evaluate(Display.Caption)
        Exit Sub
    A:
        Display.Caption = "Error"
    End Sub
Private Sub CmdEquals_Exit(ByVal Cancel As MSForms.ReturnBoolean)
'It erases the display once we select stop selecting the equals button.
    Display.Caption = Empty
End Sub
```

Ok. Let's fix the code now.

We want to tell Excel: If the display is showing the Error Message then erase it, otherwise keep it. Now, let's translate the message to some code, so Excel can understand.

```
    Private Sub CmdEquals_Exit(ByVal Cancel As
MSForms.ReturnBoolean)
        'It erases the display once we select stop selecting the equals
    button.
        If Display.Caption = "Error" Then
            Display.Caption = Empty
        Else
            Display.Caption = Display.Caption
```

```
        End If
    End Sub
```

It is exactly what we want. Try it! Type something like 3* then Equals. It displays the error message and erases it once you type something else. Now, type again 7*7 then equals. You should see 49, now type +1 It works!

It only erases the error message but keeps the numbers. Now, although it is working fine, we've added some unnecessary code: Display.Caption = Display.Caption

Let's delete it, including "else" which means "otherwise". Maybe we wouldn't notice it, but it is not professional and in a very small degree it consumes resources. Never add unnecessary code to any program you make. It will be great for you and those who use your program. If you make bad coding habits, you probably will have some future troubles when need to add pages and pages of code.

Once you delete the unnecessary code it should look like this:

```
    Private Sub CmdEquals_Exit(ByVal Cancel As
MSForms.ReturnBoolean)
        If Display.Caption = "Error" Then
            Display.Caption = Empty
        End If
    End Sub
```

It works too! You see? The code we erased was unnecessary!

To work always as a Pro, never forget these tips:

Always add comments.

Never add unnecessary code.

Always choose the correct variables

Maybe you're wondering why we didn't choose any variable for the Calculator Project. In fact, we did although we didn't declare them. All our variables are declared automatically as Variant. We needed those because this Calculator may work with billions and billions of numbers. A byte, Integer, Long, etc. wouldn't have worked in case the user needs add bigger number than the allowed by them.

Remember, if you don't declare variables they are added automatically as variants which let any value, doesn't matter how big or small it is.

The whole code for Calculator Project

For this project, the whole code in the CalculatorProject should look like this:

```
Private Sub Cmd0_Click()
   Display.Caption = Display.Caption + Cmd0.Caption
End Sub

Private Sub Cmd00_Click()
   Display.Caption = Display.Caption + Cmd00.Caption
End Sub
```

```
Private Sub cmd1_Click()
    Display.Caption = Display.Caption + cmd1.Caption
End Sub

Private Sub Cmd2_Click()
    Display.Caption = Display.Caption + Cmd2.Caption
End Sub

Private Sub Cmd3_Click()
    Display.Caption = Display.Caption + Cmd3.Caption
End Sub

Private Sub Cmd4_Click()
    Display.Caption = Display.Caption + Cmd4.Caption
End Sub

Private Sub Cmd5_Click()
    Display.Caption = Display.Caption + Cmd5.Caption
End Sub

Private Sub Cmd6_Click()
    Display.Caption = Display.Caption + Cmd6.Caption
```

```
End Sub

Private Sub Cmd7_Click()
    Display.Caption = Display.Caption + Cmd7.Caption
End Sub

Private Sub Cmd8_Click()
    Display.Caption = Display.Caption + Cmd8.Caption
End Sub

Private Sub Cmd9_Click()
    Display.Caption = Display.Caption + Cmd9.Caption
End Sub

Private Sub CmdAdd_Click()
    Display.Caption = Display.Caption + CmdAdd.Caption
End Sub

Private Sub CmdCE_Click()
    Display.Caption = Empty
End Sub

Private Sub CmdDel_Click()
```

```
    If Display.Caption <> Empty Then
        X = Len(Display.Caption) - 1
        Y = Display.Caption
        Display.Caption = Left(Y, X)
    End If
End Sub

Private Sub CmdDivision_Click()
    Display.Caption = Display.Caption + CmdDivision.Caption
End Sub

Private Sub CmdDot_Click()
    Display.Caption = Display.Caption + CmdDot.Caption
End Sub

Private Sub CmdEquals_Click()
    On Error GoTo A
        Display.Caption = Application.Evaluate(Display.Caption)
    Exit Sub
A:
        Display.Caption = "Error"
End Sub
Private Sub CmdEquals_Exit(ByVal Cancel As
```

```
MSForms.ReturnBoolean)
        If Display.Caption = "Error" Then
            Display.Caption = Empty
        End If
    End Sub

    Private Sub CmdMinus_Click()
        Display.Caption = Display.Caption + CmdMinus.Caption
    End Sub

    Private Sub CmdPercent_Click()
        Display.Caption = Display.Caption + CmdPercent.Caption
    End Sub

    Private Sub CmdTimes_Click()
        Display.Caption = Display.Caption + CmdTimes.Caption
    End Sub

    Private Sub UserForm_Terminate()
        Application.Quit
    End Sub
```

In *ThisWorkbook* the code should look like this:

```
Private Sub Workbook_Open()
    Application.Visible = False
    CalculatorProject.Show
End Sub
```

Try it! It is working great!

Adding Keyboard combination to Command Button.

There's another thing that would be great to add. Have you notice it only works by clicking on the buttons?

I think that it would be even better if we could use the keyboard instead of using all clicks only.

Let's see how to do it:

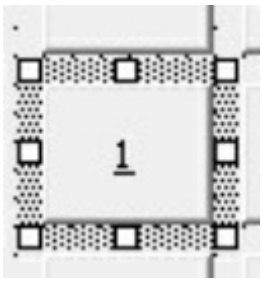
1. Open your VBA Project Form.

2. Now, let's suppose we want to add some hot key combination to Cmd1, which is the button with the number one. It is very simple.

Click on it.

3. Go to its properties.
4. In the Accelerator type any button. In this case the most appropriate one would be number 1, like in the example below:

5. After you do that, you should see the number one underlined.



6. When you see something underlined like that, not only in Excel but in all Microsoft Operating Systems, it means that if you hold Alt+ “The underlined letter or number” it will run it.
7. Try running the Calculator and Press Alt+1 to see what it does. It should type number one.
8. Do the same for all the other numbers and symbols, except for DEL, CE, 00 and =.

If you tried to add the hot key Backspace to DEL or Supr to CE

you'll notice that it is not possible. At least, not by this way.

Command Buttons Order

When you open the Calculator, you'd see that if you type TAB it will select the buttons, but probably in an unordered way. We want to give them an order. Do the following to fix that:

1. Open VBA and open your CalculatorProject.
2. Click on View TAB.
3. Click on TAB Order.
4. Rearrange the list according to the order you prefer by using the Move Up and Move Down buttons.
5. Click on Ok when you're done.

Adding Password to VBA Code.

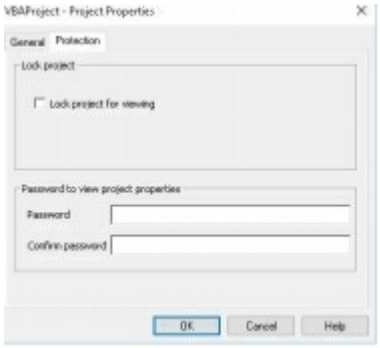
Probably you know that you can add a Password to any Excel File by following this way:

1. File TAB
2. Info
3. Protect Workbook
4. Encrypt with Password

It is a nice option, however, it is not a VBA Password. We don't want to enter a password everytime we open the calculator. The only thing we want is to protect our code from any unauthorized person who wants to change it. To do that only do the following:

1. Open VBA
2. Click on Tools
3. Click on VBAProject Properties
4. Protection TAB
5. Add a Password Lock for viewing if you want
6. Click on OK

In that way, you protect your calculator code!!



Quiz 5

What's an Userform?

a) It is a Visual Interface of an Application.

b) They are command Buttons, labels and Checkboxes.

c) It is a preloaded app which we need to edit for personal use.

How do you create a Userform?

a) Developer TAB, Insert Form

b) Visual Basic, Insert TAB, Userform

c) Visual Basic, Module, Form

What's the way to insert Excel Formulas on VBA?

a) Application.Formulas

b) Application.FormulaBarHeight

c) Application.WorksheetFunction

What Formula is made to stop Excel Visible?

a) Application.Quit

b) Application.Visible = False

c) Application.Visible = True

What are the ways to disable Macros or VBA once they are enabled?

a) Rename the file, move it from its location and in Developer TAB and Macro Security

b) Developer TAB, Macro Security only

c) There's no way to do it once they are enabled.

How to set up a Password to your VBA code?

a) Visual Basic, Tools, Options, Security, Passwords.

b) Visual Basic, Tools, VBAProject Properties, Protection, Passwords.

c) Visual Basic, File, Save as, Encrypt VBA code, Password.

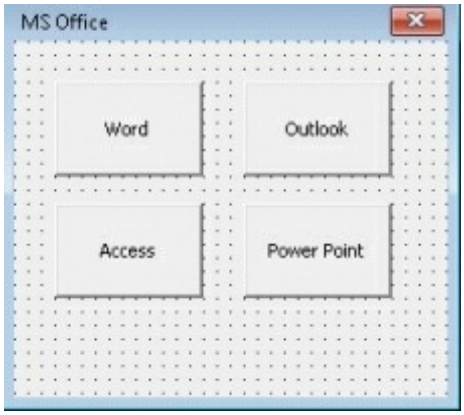
Interacting with other Applications.

Opening other apps from Excel

Microsoft Excel VBA is much more powerful than what we've seen. It can even interact with other applications, such as Microsoft Word, Access, Powerpoint, etc.

We'll just take a look about interacting with other applications from VBA.

For this practice, create the following Form:



Add the following code:

```
Private Sub cmdAccess_Click()
  Application.ActivateMicrosoftApp xlMicrosoftAccess
End Sub
```

```
Private Sub cmdOutlook_Click()
   Application.ActivateMicrosoftApp xlMicrosoftMail
End Sub

Private Sub cmdPowerPoint_Click()
Application.ActivateMicrosoftApp xlMicrosoftPowerPoint
End Sub

Private Sub cmdWord_Click()
   Application.ActivateMicrosoftApp xlMicrosoftWord
End Sub
```

It will start the applications once you click on every button.

Maybe you want to start another application. Let's try opening the Notepad.

Add this code to a new Button:

```
Private Sub CommandButton1_Click()
Dim Task As Double
  Task = Shell("notepad.exe", 1)
```

End Sub

Great! It opens the Notepad!!

Let’s do something more interactive:

Sending an Outlook e-mail from Excel:

Add this code to one button, module or Macro:

```
Dim OutlookApp As Object
Dim Email As Object
Dim Subject As String
Dim EmailAddress As String
Dim Msg As String

'Create Outlook Object
Set OutlookApp = CreateObject("Outlook.Application")

    Subject = "This is my subject"
    EmailAddress = "email@gmail.com"
```

```
  Msg = "This is the body message"

'Create email and send it:

Set Email = OutlookApp.CreateItem(0)
With Email
  .to = EmailAddress
  .Subject = Subject
  .body = Msg
  .display
  End With
```

As you see it has practically written an e-mail from Excel! Obviously, you can substitute the values for Excel Cells which could be very beneficial in case you have a data base with e-mails and messages. All you need to do is to think a little about how to do what you need.

In case you want to see how many apps VBA has available to interact with, you just need to click on Tools, References.

All the list there are applications which Excel can interact, but as you see most of them are disabled. This is just a protection feature. If they are not enough for you, you can also Browse for more.

Visual Basic for Applications is a great tool and it is one of the most powerful programming languages used in the world for Analysis. As you

learn VBA you'll make wonderful projects.

You've learned the basics to use VBA and even create Apps! Now, it is up to you how to use this knowledge.

Experiment coding, create new Projects! Do it Great!

Exercises Solutions

Answers Chapter 1

1. **How you Access the Developer TAB?**

   b) Right click on the Ribbon, Customize the Ribbon, enable the Checkbox for Developer and Accept.

Answers Chapter 2

1. **What is a Macro?**

   b) It is a shortcut which runs a recorded process.

2. **How to create a Macro?**

   c) Clicking on the Record Macro Button

3. **What is Relative References for?**

   a) It is to Record a Macro without set specific cells.

4. **How to Run a Macro?**

   a) Clicking on the Macro Button

5. **How to save a workbook with Macros?**

b) You'll get a notification, in which we should be denied and then select save as Macro-Enabled Workbook.

Answers Chapter 3

What is a Variable?

b) It is a value which changes.

What is the storage size of a byte variable and its range?

c) It stores 1 byte and goes from 0 to 256.

What is the advantage of declaring variables?

a) It consumes less RAM and makes it run smoothly.

Is it obligatory to declare variables?

a) Only if Require Variable Declaration is enabled

Msgbox returns a value depending on the button pressed:

b) It returns a value depending on the button pressed

Answers Chapter 4

What is a Private Procedure?

c) It can't be called from another part of the code

What is a Public Procedure?

a) It means that it can be called and executed from any part of the Workbook

Answers Chapter 5

What's an Userform?

a) It is a Visual Interface of an Application.

How do you create a Userform?

b) Visual Basic, Insert TAB, Userform

What's the way to insert Excel Formulas on VBA?

c) Application.WorksheetFunction

What Formula is made to stop Excel Visible?

b) Application.Visible = False

What are the ways to disable Macros or VBA once they are enabled?

a) Rename the file, move it from its location and in Developer TAB and Macro Security

How to set up a Password to your VBA code?

b) Visual Basic, Tools, VBAProject Properties, Protection, Passwords.