

## Introduction to 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.

## ❖ Microsoft Visual Basic Fundamentals

1. Access the (VisualBasic Editor)VBE Window using Alt + F11
2. Add the developer tab in the ribbon  
File→options→customize ribbon→enable developer tab

- ❖ The Microsoft Visual Basic Interface
- ❖ VBA in Visual Basic
- ❖ Macros

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.

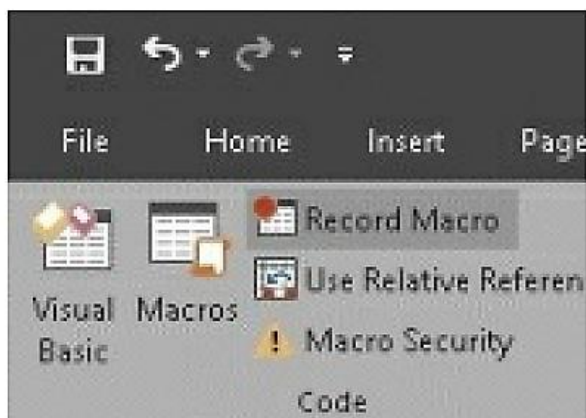
The process is the following:

- 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.

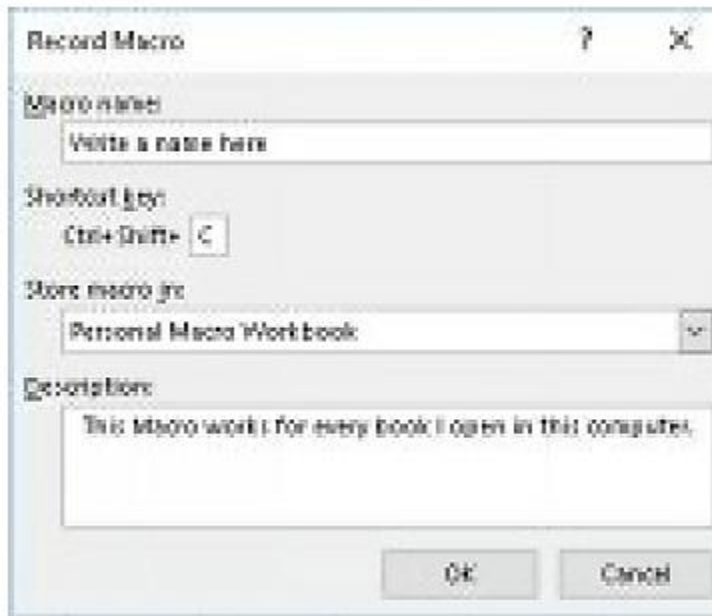
Prepwork: Create a worksheet with columns: date | month | year | your email | mobile number

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 shortcut key 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.

Note:

Add current day number  
`=day(now())`

Add current month number  
`=month(now())`

Add current year  
`=year(now())`

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

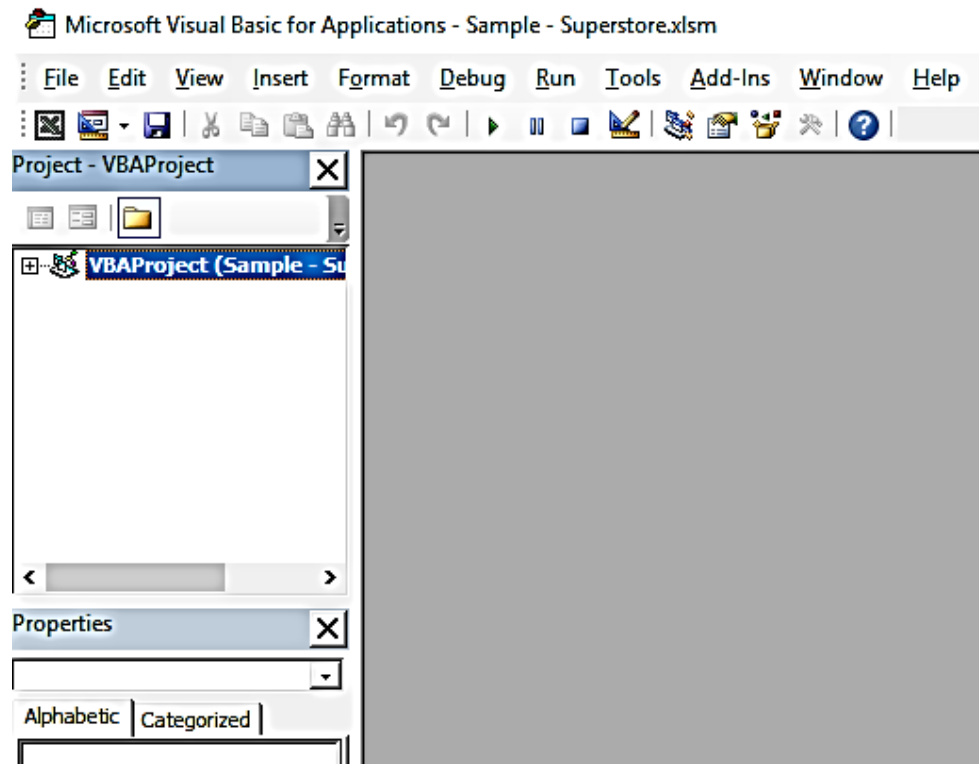
Note:

If you saved your macro in the “personal macro workbook” (making it available in all excel file in the entire computer), you might not be able to edit/delete the macro since the “personal macro workbook” is hidden by default (try it!). To resolve this, ribbon→view→unhide→PERSONAL.XLSB, then try again

❖ Writing Code (Quick sample)

### **Accessing the VBA editor (or VBE)**

- A. To access the VBE, press ALT+F11 or go to Developer tab in your ribbon (if you configured it to be displayed). You should see this window



- B. The VBE sees all your macros as well as open excel workbooks. But it can't be opened without first opening an excel workbook.

Note:

Increase font size in VBE

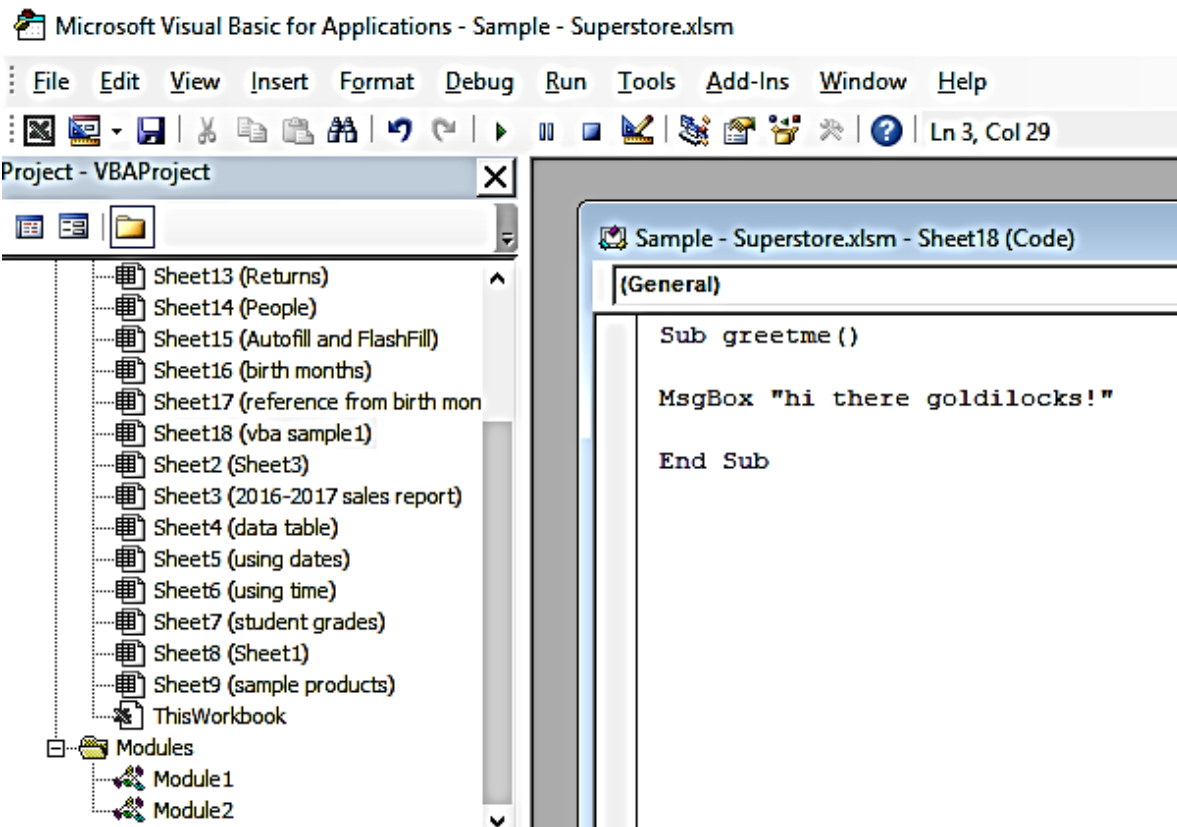
Tools→options→editor format

- C. A VBA consists of Sub procedures and / or Functions like the ff:

```
Sub addnumbers()  
  
myanswer = 20 + 20  
Msgbox "the sum is " & myanswer  
  
End Sub
```

```
Function faddnumbers(a,b)  
  
faddnumbers=a+b  
  
End Function
```

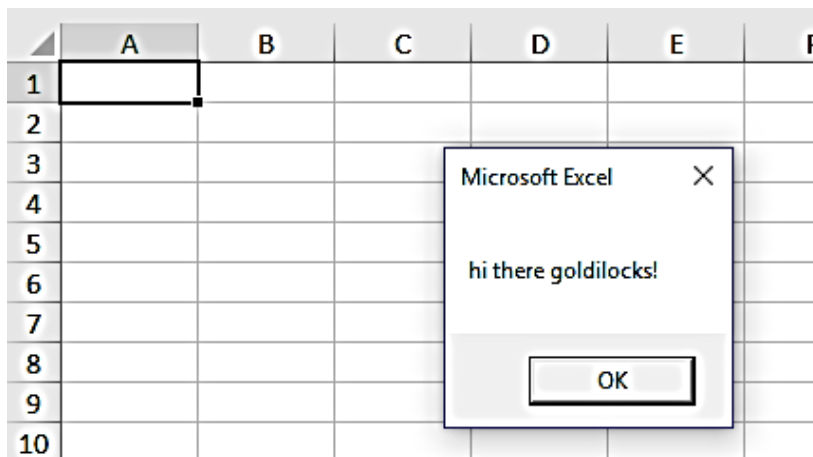
'And these can be used inside a sheet or as a module (in VBE)



Note:

- ✓ Adding custom code in VBE usually requires that an Excel Workbook must be saved as a macro-enabled workbook (xlsm)
- ✓ To assign a shortcut key to it, back to worksheet → developer tab → macros → select the macro → options

Test it by running it from VBE



**Object References**

- a. You can make reference to a workbook by:

```
Application.Workbooks("Book1.xlsx")
```

- b. You can make reference to a sheet inside a workbook by:

```
Application.Workbooks("Book1.xlsx").Worksheets("Sheet1")
```

- c. You can make reference to a cell range by:

```
Application.Workbooks("Book1.xlsx").Worksheets("Sheet1").Range("A1")
```

- d. You can make reference to an active workbook's worksheet (done automatically if you omit some parts like the ff code):

```
Worksheets("Sheet1").Range("A1")
```

**Understanding Objects**

- A. Objects have properties

```
Worksheets("Sheet1").Range("a1").Interior.Color
```

```
Worksheets("Sheet1").Range("a1").Value
```

- B. We can also set values for the properties

```
Worksheets("Sheet1").Range("a1").Interior.Color=rgb(0,0,250)
```

```
Worksheets("Sheet1").Range("a1").Value="vba placed me here"
```

- C. We can also invoke object methods

```
Worksheets("Sheet1").Range("a1").clearcontents
```

**Understanding variables and data types**

- A. Creating a variable (no specific data type)

```
myvariable = value
```

ex:

```
mybirthyear = 2007
```

```
mybirthplace = "manila"
```

- B. VBA's Built-In Data Types

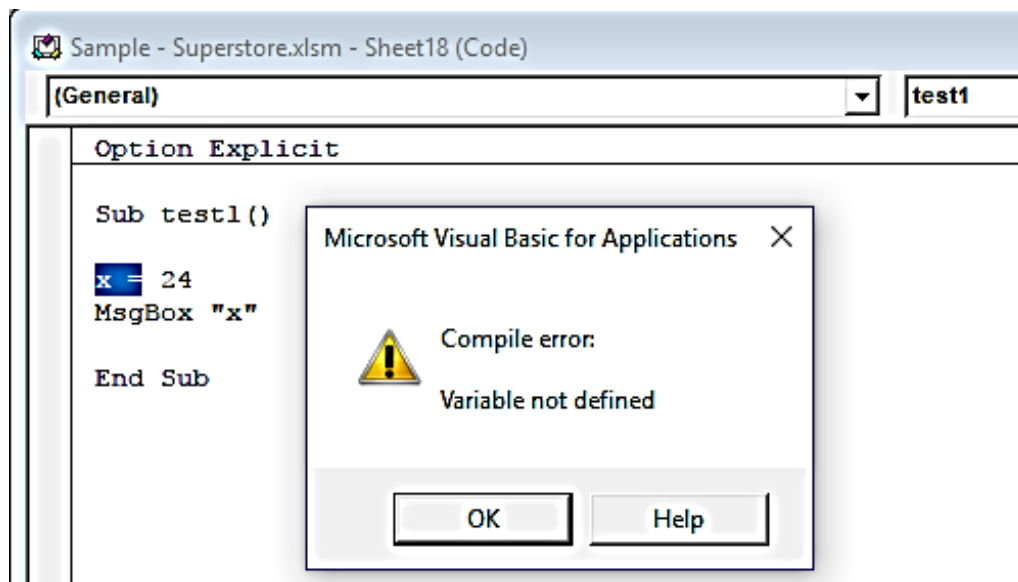
VBA's Built-In Data Types

Data Type	Bytes Used	Range of Values
Byte	1	0 to 255
Boolean	2	True or False
Integer	2	-32,768 to 32,767
Long	4	-2,147,483,648 to 2,147,483,647
Single	4	-3.40 to -1.40E-45 for negative values; 1.40E-45 to 3.40 for positive values
Double	8	-1.79E308 to -4.94E-324 for negative values; 4.94E-324 to 1.79 for positive values
Currency	8	-922,337,203,685,477 to 922,337,203,685,477

Date	8	1/1/0100 to 12/31/9999
Object	4	Any object reference
String	1 per character	Varies
Variant	Varies	Varies

- C. Before you use variables in a procedure, it's an excellent practice to declare your variables — that is, tell VBA each variable's data type. Declaring your variables makes your macro run faster and use memory more efficiently. The default data type, Variant, causes VBA to repeatedly perform time-consuming checks and reserve more memory than necessary. If VBA knows a variable's data type, it doesn't have to investigate and can reserve just enough memory to store the data.
- D. To force yourself to declare all the variables you use, include these two words as the first statement in your VBA module (When this statement is present, you won't be able to run your code if it contains any undeclared variables.):

Option Explicit



#### E. Declaring variables

Dim [variable name] as [data type]

'doesn't support group declaration like the ff:

Dim [variable1],[variable2] as integer

'the example above only sets [variable2] as integer.

'use this for multiple declaration

Dim [variable1] as integer,[variable] as integer

#### F. Variable scopes

##### Scope

Procedure only

Module only

All procedures in all modules

##### How the Variable Is Declared

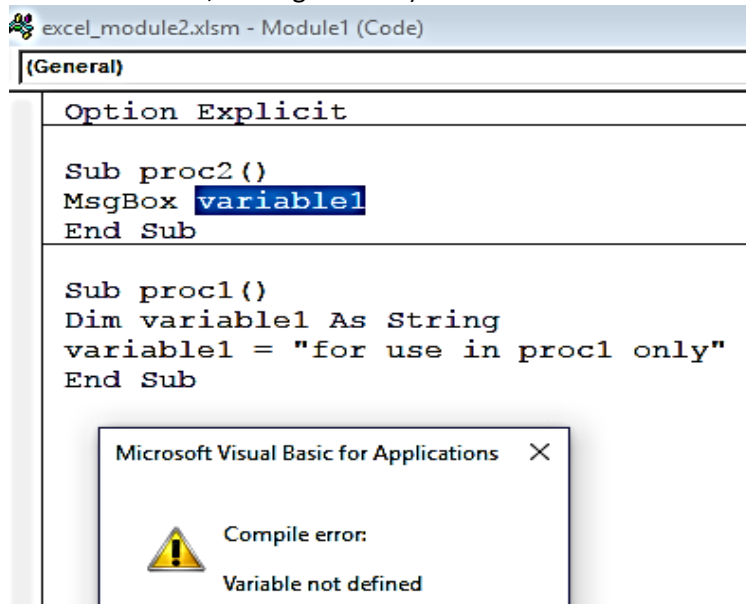
By using a Dim or a Static statement in the procedure that uses the variable

By using a Dim or a Private statement before the first Sub or Function statement in the module

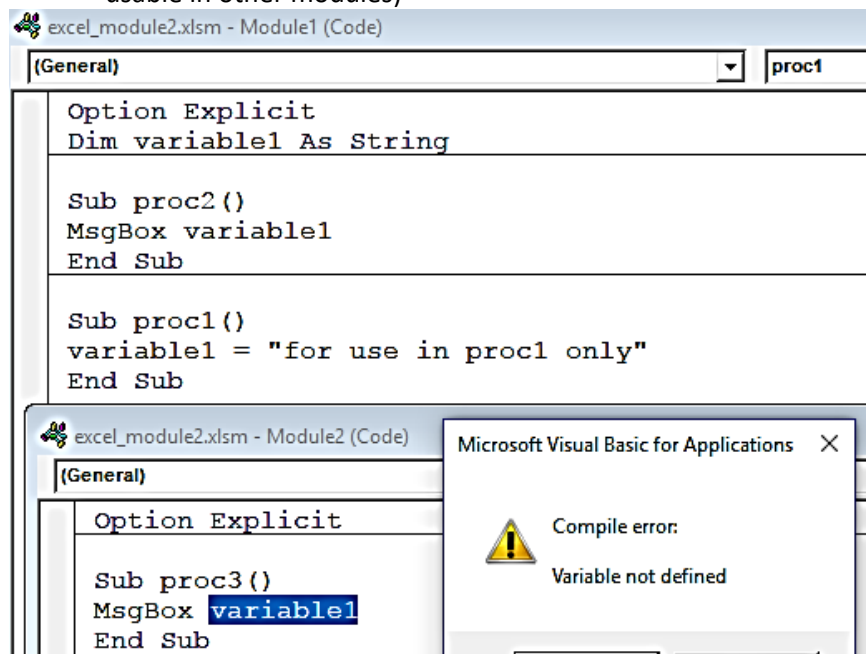
By using a Public statement before the first Sub or Function statement in a module

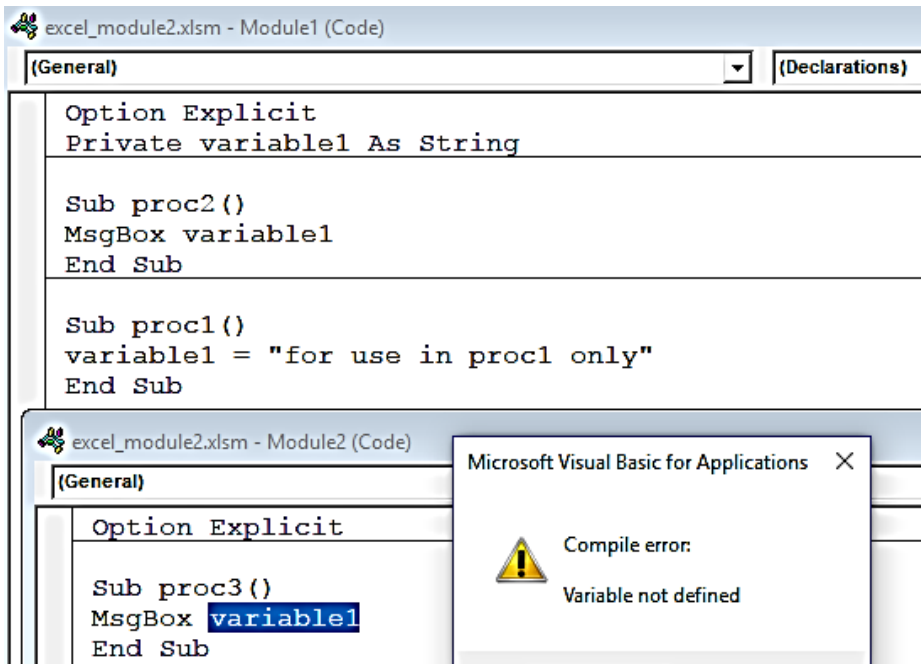
Examples:

- a. Declaring variable inside a sub or function only (although usable in other subs on the same module or in other modules, value gets reset)

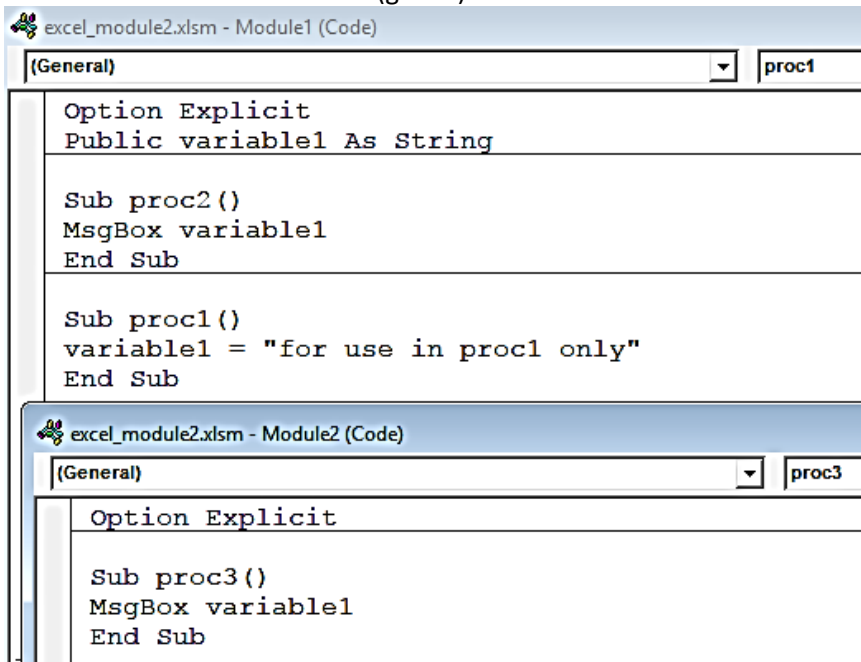


- b. Variable for all sub and function in the entire module (but the value gets reset for every sub or function. Also not usable in other modules)





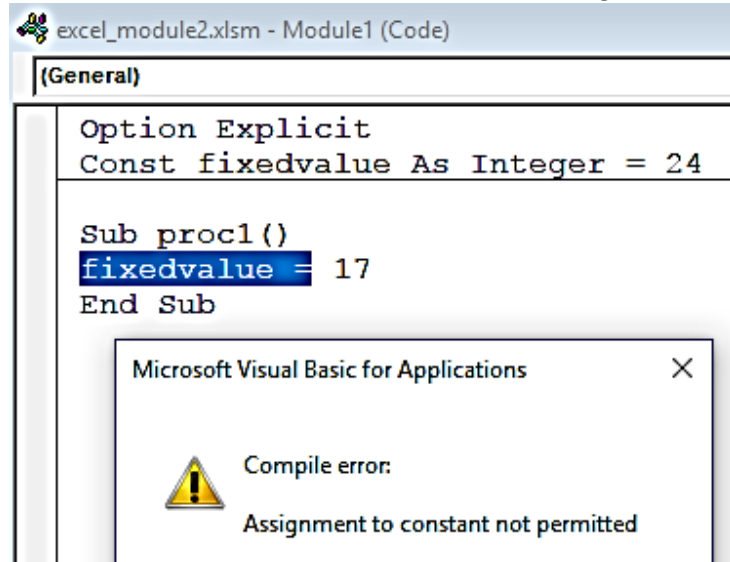
c. All module declaration (global)





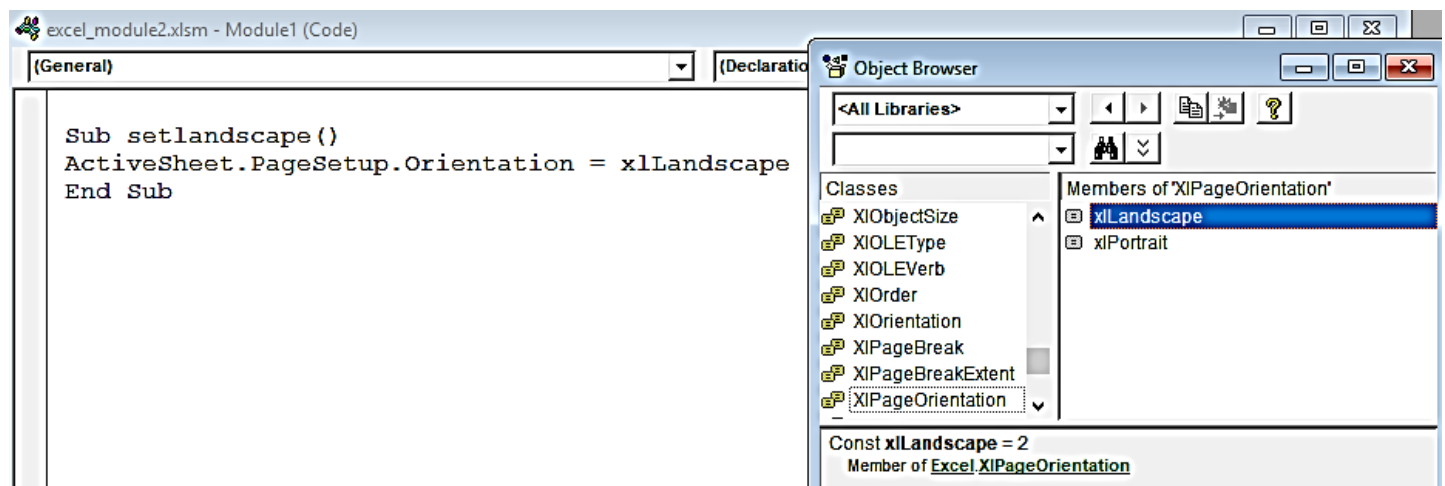
## Working with constants

'variables declared as constants cannot be re-assigned values



## Pre-made constants

'there are many predefined constants in excel functions. You may check their underlying values in the object explorer.



## Working with strings

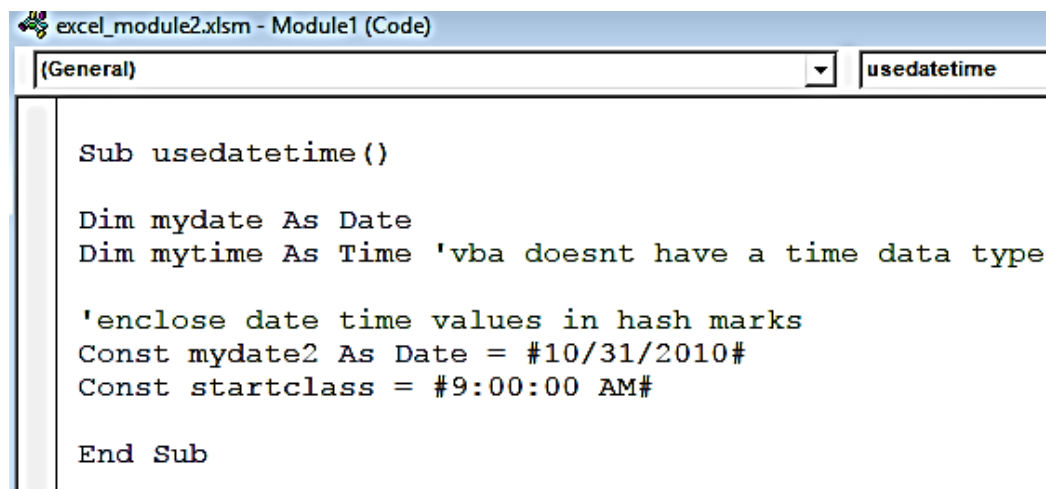
a. Declaring flexible-length string

Dim [var] as string

b. Declaring fixed-length string (example: 10 chars max)

Dim [var] as string \* 10

## Working with dates



```
Sub usedatetime()  
  
Dim mydate As Date  
Dim mytime As Time 'vba doesnt have a time data type  
  
'enclose date time values in hash marks  
Const mydate2 As Date = #10/31/2010#  
Const startclass = #9:00:00 AM#  
  
End Sub
```

## Using Assignment Statements

```
x = 1  
x = x + 1  
x = (y * 2) / (z * 2)  
HouseCost = 375000  
FileOpen = True  
Range("TheYear").Value = 2013
```

Function Operator	Symbol
Addition	+
Multiplication	*
Division	/
Subtraction	-
Exponentiation	^
String concatenation	&
Integer division (the result is always an integer) \	
Modulo arithmetic (returns the remainder of a division operation)	Mod

## Working with Arrays

- An array is a variable that holds multiple values assigned to array indices. An array must be declared before use.
- To declare an array:

```
Dim array1(1 to 10) as string  
  
'if start index not defined, it start with 0  
Dim array2(10) as string
```

c. To assign values to array

```
array1(1) = one  
array1(2) = two
```

d. To create a dynamic array

```
'declare an empty array first in the global declarations  
Dim dynamicarray() as string
```

```
'then once you have recomputed the possible size (example: using counta) you can then resize the array using redim.  
'Using +1 is a good idea so you will always have an empty array index at the end, but note that the last array index will  
'always be empty.  
Redim dynamicarray(newsize + 1)
```

e. Declaring multi-dimensional array

```
'this creates 3 storage index for each of the first set of index (total of 9 indices)  
Dim multidimarray(1 to 3,1 to 3) as string
```

f. Assign value to multi-dimensional array

```
multidimarray(1,1) = A
```

## VBA Operators and Operands

### ❖ VBA Operators

An **Operator** can be defined using a simple expression - 4 + 5 is equal to 9. Here, 4 and 5 are called **operands** and + is called **operator**. VBA supports following types of operators –

- ▣ Arithmetic Operators
- ▣ Comparison Operators
- ▣ Logical (or Relational) Operators
- ▣ Concatenation Operators

## The Arithmetic Operators

Following arithmetic operators are supported by VBA.

Assume variable A holds 5 and variable B holds 10, then –

Show Examples [↗](#)

Operator	Description	Example
+	Adds the two operands	A + B will give 15
-	Subtracts the second operand from the first	A - B will give -5
*	Multiplies both the operands	A * B will give 50
/	Divides the numerator by the denominator	B / A will give 2
%	Modulus operator and the remainder after an integer division	B % A will give 0
^	Exponentiation operator	B ^ A will give 100000

## The Comparison Operators

There are following comparison operators supported by VBA.

Assume variable A holds 10 and variable B holds 20, then –


Show Examples [↗](#)

Operator	Description	Example
=	Checks if the value of the two operands are equal or not. If yes, then the condition is true.	(A = B) is False.
<>	Checks if the value of the two operands are equal or not. If the values are not equal, then the condition is true.	(A <> B) is True.
>	Checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition is true.	(A > B) is False.
<	Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition is true.	(A < B) is True.
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition is true.	(A >= B) is False.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition is true.	(A <= B) is True.

## The Logical Operators

Following logical operators are supported by VBA.

Assume variable A holds 10 and variable B holds 0, then –


Show Examples 

Operator	Description	Example
AND	Called Logical AND operator. If both the conditions are True, then the Expression is true.	a<>0 AND b<>0 is False.
OR	Called Logical OR Operator. If any of the two conditions are True, then the condition is true.	a<>0 OR b<>0 is true.
NOT	Called Logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	NOT(a<>0 OR b<>0) is false.
XOR	Called Logical Exclusion. It is the combination of NOT and OR Operator. If one, and only one, of the expressions evaluates to be True, the result is True.	(a<>0 XOR b<>0) is true.

## The Concatenation Operators

Following Concatenation operators are supported by VBA.

Assume variable A holds 5 and variable B holds 10 then –

Show Examples 

Operator	Description	Example
+	Adds two Values as Variable. Values are Numeric	A + B will give 15
&	Concatenates two Values	A & B will give 510

Assume variable A = "Microsoft" and variable B = "VBScript", then –

Operator	Description	Example
+	Concatenates two Values	A + B will give MicrosoftVBScript
&	Concatenates two Values	A & B will give MicrosoftVBScript

**Note** – Concatenation Operators can be used for both numbers and strings. The output depends on the context, if the variables hold numeric value or string value.

**12nn-1pm**

Lunch Break

**1pm-3pm**

## Using Labels

```
Sub linelabels()
```

```
Dim myinput As Integer
```

```
myinput = 14
```

```
If myinput = 1 Then GoTo ifone
```

```
If myinput <> 1 Then GoTo notone
```

```
ifone:
```

```
MsgBox "one"
```

```
Exit Sub
```

```
notone:
```

```
MsgBox "not one"
```

```
Exit Sub
```

```
End Sub
```

## Using the Range Object

### a. Active worksheet

Range("A1:C5")	'cell range
Range("K9")	'single cell address
Range("PriceList")	'named range in excel
Range("A:A")	'entire column
Range("3:3")	'entire row

### b. Other worksheet

```
Worksheets("Sheet1").Range("A1:C5")
```

### c. Other workbook

```
Workbooks("Budget.xlsx").Worksheets("Sheet1").Range("A1:C5")
```

### d. Multiple non-contiguous section

```
Range("A1:B8,D9:G16")
```

### **The Cells property**

'takes in two arguments, row and column. The intersection can be used to point to a cell

```
Worksheets("Sheet2").Cells(2, 3)
```

'can also be used to highlight a cell range

```
Range(cells(1,1),cells(10,8))
```

'cells property has an advantage over range when you use variables and logic to select cells.

### **The Offset property**

'two rows below A1 and three cells to the right of A1, this refers to D3

```
Range("A1").Offset(2,3)
```

'can also accept negative values for the arguments, the arguments still reflect rows and columns(in order).

'the following example refers to A1

```
Range("C2").Offset(-1,-3)
```

'to use relative references, offset can be used along with the ActiveCell object

'example: write the date on the right of the active cell

```
ActiveCell.Offset(0,1)=Date
```

### **The Value property**

'displays a message box that shows the value in cell A1 on Sheet1

```
MsgBox Worksheets("Sheet1").Range("A1").Value
```

'without assigning to a variable, you can read the Value property only for a single-cell range object. The 'following statement generates an error:

```
MsgBox Worksheets("Sheet1").Range("A1:C3").Value
```

'change the Value property for a range of any size.

```
Worksheets("Sheet1").Range("A1:C3").Value = 123
```

'Value is the default property for a Range object.

```
Range("A1").Value = 75
```

```
Range("A1") = 75
```

'assign values in a multi-cell range to a variable

'the following example displays the data in C1

```
Dim range1 as variant
```

```
Range1=Range("A1:C3").Value
```

```
Msgbox Range1(1,3)
```

### **The Text property**

'returns the formatted display value in a cell (not the raw logical value).

```
Worksheets("Sheet1").Range("A1").Text
```

## **The Count property**

Dim a As Integer, b As Integer, c As Integer

'returns all cells

a = Range("A1:B3").Count

'returns all non-blank cells

b = Range("A1:B3").SpecialCells(2).Count

'1-numbers

'2-textvalues

'4-logical

'returns all non-blank cells

c = Application.WorksheetFunction.CountA(Range("a1:b3"))

## **The Column and Row properties**

'The Column property returns the column number of a single-cell range. (the ff example returns 6)

MsgBox Sheets("Sheet1").Range("F3").Column

'The Row property, returns the row number of a single-cell range. (the ff example returns 3)

MsgBox Sheets("Sheet1").Range("F3").Row

## **The Address property**

'displays the cell address for a Range object as an absolute reference (a dollar sign before the column letter and before the row number) (read-only property)

MsgBox Range(Cells(1, 1), Cells(5, 5)).Address                      'this returns \$A\$1:\$E\$5

## **The HasFormula property**

'this is a Boolean read-only property that returns true if the selected range has a formula and false if none

Dim FormulaTest As Boolean

FormulaTest = Range("A1").HasFormula

'when using a range that has cells with formula and non-formula, the ff will generate an error

Dim FormulaTest As Boolean

FormulaTest = Range("A1:A3").HasFormula

'use the following to resolve this

Sub CheckForFormulas()

Dim FormulaTest As Variant

FormulaTest = Range("A1:A2").HasFormula

If TypeName(FormulaTest) = "Null" Then

MsgBox "Mixed!"

Else

MsgBox FormulaTest

End If

End Sub



### **The Font property**

```
Range("A1:B3").Font.Bold = True  
Range("A1:B3").Font.Color = vbRed  
Range("A1:B3").Font.Italic = True  
Range("A1:B3").Font.FontStyle = "arial"  
Range("A1:B3").Font.Underline = True
```

### **The Interior property**

```
Range("A1:B3").Interior.Color = rgb(0,200,0)
```

### **The Formula property**

'represents the formula in a cell  
Range("A13").Formula = "=SUM(A1:A12)"

'if you want to concatenate a string(with quotes) to the formula, the following will display an error:  
=SUM(A1:A12)&" Stores"

'to fix, replace all single double quotes with two double quotes  
"=SUM(A1:A12)&"" Stores"""

### **The NumberFormat property**

```
Columns("A:A").NumberFormat = "0.00%"
```

'more number formats can easily be seen in format cells dialog box

### **The Select method**

'selects a range on the active worksheet:  
Range("A1:C12").Select

'if Sheet1 contains the range you want to select, use the following statements to select the range:  
Sheets("Sheet1").Activate  
Range("A1:C12").Select

'This statement activates Sheet1 and then selects the range:  
Application.Goto Sheets("Sheet1").Range("A1:C12")

### **The Copy and Paste methods**

```
'copies range A1:A12 and pastes it to the same worksheet, beginning at cell C1:  
Sub CopyRange()  
Range("A1:A12").Select  
Selection.Copy  
Range("C1").Select  
ActiveSheet.Paste  
End Sub
```

'the ff can also be used  
Range("A1:A12").Copy Range("C1")

### **The Clear method**

'removes formatting only  
Columns("D:D").ClearFormats

'remove contents but leaves formatting  
Columns("D:D").ClearContents

'removes contents and formatting  
Columns("D:D").Clear

### **The Delete method**

'When you delete a range, Excel shifts the remaining cells around to fill up the range you deleted.  
Rows("6:6").Delete

'When you delete a range that's not a complete row or column, Excel needs to know how to shift the cells.  
'The following statement deletes a range and then fills the resulting gap by shifting the other cells to the left:  
Range("C6:C10").Delete xlToLeft

'The Delete method uses an argument that indicates how Excel should shift the remaining cells. In this case, I use a built-in constant (xlToLeft) for the argument. I could also use xlUp, another named constant.

### **VBA functions**

Here are some of popular VBA built-in commands

Function	What It Does
Abs	Returns a number's absolute value.
Array	Returns a variant containing an array.
Choose	Returns a value from a list of items.
Chr	Converts an ANSI value to a string.
CurDir	Returns the current path.
Date	Returns the current system date.
DateAdd	Returns a date to which a specified time interval has been added — for example, one month from a particular date.
DateDiff	Returns an integer showing the number of specified time intervals between two dates — for example, the number of months between now and your birthday.
DatePart	Returns an integer containing the specified part of a given date —for example, a date's day of the year.
DateSerial	Converts a date to a serial number.
DateValue	Converts a string to a date.
Day	Returns the day of the month from a date value.
Dir	Returns the name of a file or directory that matches a pattern.
Err	Returns the error number of an error condition.
Error	Returns the error message that corresponds to an error number.
Exp	Returns the base of the natural logarithm (e) raised to a power.

FileLen	Returns the number of bytes in a file.
Fix	Returns a number's integer portion.
Format	Displays an expression in a particular format.
GetSetting	Returns a value from the Windows registry.
Hour	Returns the hours portion of a time.
InputBox	Displays a box to prompt a user for input.
InStr	Returns the position of a string within another string.
InStrRev	Returns the position of a string within another, from the end of a string.
Int	Returns the integer portion of a number.
IsArray	Returns True if a variable is an array.
IsDate	Returns True if an expression is a date.
IsEmpty	Returns True if a variable has not been initialized.
IsError	Returns True if an expression is an error value.
IsMissing	Returns True if an optional argument was not passed to a procedure.
IsNull	Returns True if an expression contains no valid data.
IsNumeric	Returns True if an expression can be evaluated as a number.
LBound	Returns the smallest subscript for a dimension of an array.
LCase	Returns a string converted to lowercase.
Left	Returns a specified number of characters from the left of a string.
Len	Returns the number of characters in a string.
Mid	Returns a specified number of characters from a string.
Minute	Returns the minutes portion of a time value.
Month	Returns the month from a date value.
MsgBox	Displays a message box and (optionally) returns a value.
Now	Returns the current system date and time.
Replace	Replaces a substring in a string with another substring.
RGB	Returns a numeric RGB value representing a color.
Right	Returns a specified number of characters from the right of a string.
Rnd	Returns a random number between 0 and 1.
Second	Returns the seconds portion of a time value.
Shell	Runs an executable program.
Space	Returns a string with a specified number of spaces.
Split	Splits a string into parts, using a delimiting character.
Sqr	Returns a number's square root.
String	Returns a repeating character or string.
Time	Returns the current system time.
Timer	Returns the number of seconds since midnight.
TimeSerial	Returns the time for a specified hour, minute, and second.
TimeValue	Converts a string to a time serial number.
Trim	Returns a string without leading or trailing spaces.
TypeName	Returns a string that describes a variable's data type.
UBound	Returns the largest available subscript for an array's dimension.
UCase	Converts a string to uppercase.
Val	Returns the numbers contained in a string.
Weekday	Returns a number representing a day of the week.
Year	Returns the year from a date value.

Examples:

```
Msgbox "today is " & Date  
Msgbox "string length of data in A1: " & Len(Range("A1"))  
Msgbox "the current month is " & MonthName(Month(Date))  
Ucase(Range("A1"))
```

## **Using Worksheet Functions in VBA**

'if there are functions in excel that you prefer to use, you can invoke the worksheetfunction object

Examples:

'adding cell range using excel's sum

```
Dim mytotal as double  
Mytotal = Application.WorksheetFunction.Sum(Range(A:A))  
'the ff also gets the same answer  
WorksheetFunction.Sum(Range("A:A"))  
Application.Sum(Range(A:A))
```

'getting the maximum value in a range

```
Dim maxval as double  
maxval = WorksheetFunction.Max(Range("A:A"))
```

'getting the second largest value

```
Dim secondbiggest as double  
Secondbiggest = Large(Range("A:A"),2)
```

'sample using vlookup

```
Sub GetPrice()  
Dim PartNum As Variant  
Dim Price As Double  
PartNum = InputBox("Enter the Part Number")  
Sheets("Prices").Activate  
Price = WorksheetFunction.VLOOKUP(PartNum, Range("PriceList"), 2, False)  
MsgBox PartNum & " costs " & Price  
End Sub
```

## **Entering worksheet functions**

'you cannot copy-paste functions from excel to the vbe. You have to do your vba codes by hand but the following ,ight come in handy:

- Enable auto-list member in options (VBE tools→options→edito tab)
- Type either worksheetfunction followed by a dot(.) then wait and view the list of available functions
- Ctrl + shift after typing an object to view its functions or child objects

## **Custom Functions**

'you can create your own functions in vba

Example:

```
Function MultiplyTwo(num1, num2) As Double
MultiplyTwo = num1 * num2
End Function

Sub ShowResult()
Dim n1 As Double, n2 As Double
Dim Result As Double
n1 = 230
n2 = 2
Result = MultiplyTwo(n1, n2)
MsgBox Result
End Sub
```

## **The GoTo Statement**

'a programming construct that lets you jump to a line label (usually for 'on error' events but try to avoid as much as you can as it can result in a messy code)

Example:

```
Sub CheckUser()

UserName = InputBox("Enter Your Name: ")
If UserName <> "John Doe" Then GoTo WrongName
MsgBox ("Welcome Steve...")
Exit Sub

WrongName:
MsgBox "Sorry. Only John Doe can run this."

End Sub
```

## **The If-Then structure**

'a programming construct that handles and delivers results for conditional statements

'if-then-else sample

```
Dim checkgrade as double
checkgrade = Inputbox("enter grade to check")
if checkgrade >= 75 then
msgbox "passed"
else
msgbox "failed"
end if
```

'if-then-elseif-else

```
Dim Msg As String
If Time < 0.5 Then
Msg = "Morning"
ElseIf Time >= 0.5 And Time < 0.75 Then
Msg = "Afternoon"
Else
Msg = "Evening"
End If
MsgBox "Good " & Msg
```

## **The Select Case structure**

'an alternative to the if-then-else structure

```
Dim Quantity As Integer
Dim Discount As Double
Quantity = InputBox("Enter Quantity: ")

Select Case Quantity
Case 0 To 24
Discount = 0.1
Case 25 To 49
Discount = 0.15
Case 50 To 74
Discount = 0.2
Case Is >= 75
Discount = 0.25
End Select

MsgBox "Discount: " & Discount
```

## **For-Next loops**

a. For next loop with a single step increment

Example:

```
Sub loopovercells()
Dim cnt As Integer
For cnt = 1 To 10
Cells(cnt, 1).Value = "loop " & cnt
Next cnt
End Sub
```

- b. For next loop with step increment

Example:

```
Sub loopovercells()  
Dim cnt As Integer  
For cnt = 1 To 10 step 2  
    Cells(cnt, 1).Value = "loop " & cnt  
Next cnt  
End Sub
```

### **Do-While loop**

'for some cases that loops may handle more than just number brackets, you may want to try the ff:

'this loop evaluates the condition first before executing the loop

'example 1:

```
Dim i As Integer  
i = 1  
Do While i <= 10  
    Cells(i, 1).Value = "test"  
    i = i + 1  
Loop
```

'example 2 (create a row with numbers in your spreadsheet for this demo):

```
Do While ActiveCell.Value <> Empty  
ActiveCell.Value = ActiveCell.Value * 2  
ActiveCell.Offset(0, 1).Select
```

### **Do-Until loop**

'same as the do-while loop, this loop can be used to iterate over a non-numeric condition

'example 1:

```
Dim i As Integer  
i = 1  
Do until i = 10  
    Cells(i, 1).Value = "test"  
    i = i + 1  
Loop
```

'example 2 (create a row with numbers in your spreadsheet for this demo):

```
Do Until IsEmpty(ActiveCell.Value)  
ActiveCell.Value = ActiveCell.Value * 2  
ActiveCell.Offset(1, 0).Select  
Loop
```

## Looping through a Collection

'VBA supports yet another type of looping — looping through each object in a collection of objects. Recall that a 'collection consists of a number of objects of the same type.

'For example, Excel has a collection of all open workbooks (the Workbooks collection), and each workbook has a 'collection of worksheets (the Worksheets collection). When you need to loop through each object in a collection, use 'the For Each-Next structure.

'Example loops through each worksheet in the active workbook and deletes the worksheet if it's empty:

Sub DeleteEmptySheets()

Dim WkSht As Worksheet

Application.DisplayAlerts = False

For Each WkSht In ActiveWorkbook.Worksheets

    If WorksheetFunction.CountA(WkSht.Cells) = 0 Then

        WkSht.Delete

    End If

Next WkSht

Application.DisplayAlerts = True

End Sub

'example 2 (loop to hide all worksheets in the active workbook, except the active sheet).

Sub HideSheets()

Dim Sht As Worksheet

For Each Sht In ActiveWorkbook.Worksheets

    If Sht.Name <> ActiveSheet.Name Then

        Sht.Visible = xlSheetHidden

    End If

Next Sht

End Sub

'example 3 (unhides all worksheets)

Sub UnhideSheets()

Dim Sht As Worksheet

For Each Sht In ActiveWorkbook.Worksheets

    Sht.Visible = xlSheetVisible

Next Sht

End Sub



## Working with Arrays

- a. An array is a variable that holds multiple values assigned to array indices. An array must be declared before use.
- b. To declare an array:

```
Dim array1(1 to 10) as string
```

'if start index not defined, it start with 0

```
Dim array2(10) as string
```

- c. To assign values to array

```
array1(1) = one
```

```
array1(2) = two
```

- d. To create a dynamic array

'declare an empty array first in the global declarations

```
Dim dynamicarray() as string
```

'then once you have recomputed the possible size (example: using counta) you can then resize the array using redim.  
'Using +1 is a good idea so you will always have an empty array index at the end, but note that the last array index will  
'always be empty.

```
Redim dynamicarray(newsize + 1)
```

- e. Declaring multi-dimensional array

'this creates 3 storage index for each of the first set of index (total of 9 indices)

```
Dim multidimarray(1 to 3,1 to 3) as string
```

- f. Assign value to multi-dimensional array

```
multidimarray(1,1) = A
```

sample excel range to array

```
Sub testarray()
```

'declare an array

```
Dim names() As Variant
```

```
Dim eachname As String
```

```
Dim i As Integer
```

```
names = WorksheetFunction.Transpose(Worksheets("sheet1").Range("a2:a5"))
```

```
For i = LBound(names) To UBound(names)
```

```
eachname = eachname & names(i) & vbCrLf
```

```
Next i
```

```
MsgBox eachname
```

```
End Sub
```

Sample excel range that gets last row with value

```
Sub testarray()  
  
'declare an array  
Dim names()  
Dim eachname As String  
Dim i As Long  
  
'get cells with values only  
Dim lastrow  
lastrow = ActiveSheet.Columns(1).SpecialCells(xlLastCell).Row  
names = WorksheetFunction.Transpose(Worksheets("sheet1").Range(Cells(2, 1), Cells(lastrow, 1)))  
  
For i = LBound(names) To UBound(names)  
    eachname = eachname & names(i) & vbCrLf  
Next i  
MsgBox eachname  
  
End Sub
```

## Understanding Objects

A. Objects have properties

```
Worksheets("Sheet1").Range("a1").Interior.Color  
Worksheets("Sheet1").Range("a1").Value
```

B. We can also set values for the properties

```
Worksheets("Sheet1").Range("a1").Interior.Color=rgb(0,0,250)  
Worksheets("Sheet1").Range("a1").Value="vba placed me here"
```

C. We can also invoke object methods

```
Worksheets("Sheet1").Range("a1").clearcontents
```

## Using the Range Object

a. Active worksheet

Range("A1:C5")	'cell range
Range("K9")	'single cell address
Range("PriceList")	'named range in excel
Range("A:A")	'entire column
Range("3:3")	'entire row

b. Other worksheet

```
Worksheets("Sheet1").Range("A1:C5")
```

c. Other workbook

```
Workbooks("Budget.xlsx").Worksheets("Sheet1").Range("A1:C5")
```

d. Multiple non-contiguous section

```
Range("A1:B8,D9:G16")
```

e. Get values from another workbook (uses code to open and close the workbook)

```
Sub getvalue()
```

```
Dim myval As String
```

```
Dim wb As Workbook
```

```
Set wb = Workbooks.Open("c:\users\core360\desktop\book3.xlsm")
```

```
myval = Workbooks("Book3.xlsm").Worksheets("sheet1").Range("a1").Value
```

```
wb.Close
```

```
MsgBox (myval)
```

```
End Sub
```

### **The Cells property**

'takes in two arguments, row and column. The intersection can be used to point to a cell

```
Worksheets("Sheet2").Cells(2, 3)
```

'can also be used to highlight a cell range

```
Range(cells(1,1),cells(10,8))
```

'cells property has an advantage over range when you use variables and logic to select cells.

### **The Offset property**

'two rows below A1 and three cells to the right of A1, this refers to D3

```
Range("A1").Offset(2,3)
```

'can also accept negative values for the arguments, the arguments still reflect rows and columns(in order).

'the following example refers to A1

```
Range("C2").Offset(-1,-3)
```

'to use relative references, offset can be used along with the ActiveCell object

'example: write the date on the right of the active cell

```
ActiveCell.Offset(0,1)=Date
```

### **The Value property**

'displays a message box that shows the value in cell A1 on Sheet1

```
MsgBox Worksheets("Sheet1").Range("A1").Value
```

'without assigning to a variable, you can read the Value property only for a single-cell range object. The 'following statement generates an error:

```
MsgBox Worksheets("Sheet1").Range("A1:C3").Value
```

'change the Value property for a range of any size.

```
Worksheets("Sheet1").Range("A1:C3").Value = 123
```

'Value is the default property for a Range object.

```
Range("A1").Value = 75
```

```
Range("A1") = 75
```

'assign values in a multi-cell range to a variable

'the following example displays the data in C1

```
Dim range1 as variant
```

```
Range1=Range("A1:C3").Value
```

```
Msgbox Range1(1,3)
```

### **The Text property**

'returns the formatted display value in a cell (not the raw logical value).

```
Worksheets("Sheet1").Range("A1").Text
```

### **The Count property**

```
Dim a As Integer, b As Integer, c As Integer
```

'returns all cells

```
a = Range("A1:B3").Count
```

'returns all non-blank cells

```
b = Range("A1:B3").SpecialCells(2).Count
```

'1-numbers

'2-textvalues

'4-logical

'returns all non-blank cells

```
c = Application.WorksheetFunction.CountA(Range("a1:b3"))
```

### **The Column and Row properties**

'The Column property returns the column number of a single-cell range. (the ff example returns 6)

```
MsgBox Sheets("Sheet1").Range("F3").Column
```

'The Row property, returns the row number of a single-cell range. (the ff example returns 3)

```
MsgBox Sheets("Sheet1").Range("F3").Row
```

### **The Address property**

'displays the cell address for a Range object as an absolute reference (a dollar sign before the column letter and before

'the row number) (read-only property)

```
MsgBox Range(Cells(1, 1), Cells(5, 5)).Address
```

'this returns \$A\$1:\$E\$5

### **The HasFormula property**

'this is a Boolean read-only property that returns true if the selected range has a formula and false if none

```
Dim FormulaTest As Boolean
```

```
FormulaTest = Range("A1").HasFormula
```

'when using a range that has cells with formula and non-formula, the ff will generate an error

```
Dim FormulaTest As Boolean
FormulaTest = Range("A1:A3").HasFormula
```

```
'use the following to resolve this
Sub CheckForFormulas()
Dim FormulaTest As Variant
FormulaTest = Range("A1:A2").HasFormula
If TypeName(FormulaTest) = "Null" Then
MsgBox "Mixed!"
Else
MsgBox FormulaTest
End If
End Sub
```

### **The Font property**

```
Range("A1:B3").Font.Bold = True
Range("A1:B3").Font.Color = vbRed
Range("A1:B3").Font.Italic = True
Range("A1:B3").Font.FontStyle = "arial"
Range("A1:B3").Font.Underline = True
```

### **The Interior property**

```
Range("A1:B3").Interior.Color = rgb(0,200,0)
```

### **The Formula property**

```
'represents the formula in a cell
Range("A13").Formula = "=SUM(A1:A12)"

'if you want to concatenate a string(with quotes) to the formula, the following will display an error:
=SUM(A1:A12)&" Stores"

'to fix, replace all single double quotes with two double quotes
"=SUM(A1:A12)&" Stores""
```

### **The NumberFormat property**

```
Columns("A:A").NumberFormat = "0.00%"

'more number formats can easily be seen in format cells dialog box
```

### **The Select method**

```
'selects a range on the active worksheet:
Range("A1:C12").Select

'if Sheet1 contains the range you want to select, use the following statements to select the range:
Sheets("Sheet1").Activate
Range("A1:C12").Select
```

'This statement activates Sheet1 and then selects the range:  
Application.Goto Sheets("Sheet1").Range("A1:C12")

## **The Copy and Paste methods**

```
Sub Excel_VBA_Copy_Range_to_Another_Sheet_with_FormattingAndColumnWidths()  
Range("A:E").Copy Destination:=Sheets("AnotherSheet").Range("a1")  
End Sub
```

'the ff can also be used  
Range("A1:A12").Copy Range("C1")

Copy including column width

```
Sub Excel_VBA_Copy_Range_to_Another_Sheet_with_FormattingForEachColumn()  
  
Range("A1:E21").Copy Destination:=Sheets("AnotherSheet").Range("a1")  
colCntr = 0  
  
For Each col In Range("A1:E21").Columns  
Sheets("AnotherSheet").Range("A1").Offset(1, colCntr).ColumnWidth = col.ColumnWidth  
colCntr = colCntr + 1  
Next  
End Sub
```

Copy values only without formatting

```
Range("A1:E21").Copy  
  
Sheets("AnotherSheet").Range("A1").PasteSpecial _  
Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks:=False, Transpose:=False
```

## **The Clear method**

```
'removes formatting only  
Columns("D:D").ClearFormats  
  
'remove contents but leaves formatting  
Columns("D:D").ClearContents  
  
'removes contents and formatting  
Columns("D:D").Clear
```

## **The Delete method**

'When you delete a range, Excel shifts the remaining cells around to fill up the range you deleted.  
Rows("6:6").Delete

'When you delete a range that's not a complete row or column, Excel needs to know how to shift the cells.  
'The following statement deletes a range and then fills the resulting gap by shifting the other cells to the left:

```
Range("C6:C10").Delete xlToLeft
```

'The Delete method uses an argument that indicates how Excel should shift the remaining cells. In this case, I use a built-in constant (xlToLeft) for the argument. I could also use xlUp, another named constant.

### **For-Next loops**

- a. For next loop with a single step increment

Example:

```
Sub loopovercells()  
Dim cnt As Integer  
For cnt = 1 To 10  
    Cells(cnt, 1).Value = "loop " & cnt  
Next cnt  
End Sub
```

- b. For next loop with step increment

Example:

```
Sub loopovercells()  
Dim cnt As Integer  
For cnt = 1 To 10 step 2  
    Cells(cnt, 1).Value = "loop " & cnt  
Next cnt  
End Sub
```

### **Do-While loop**

'for some cases that loops may handle more than just number brackets, you may want to try the ff:

'this loop evaluates the condition first before executing the loop

'example 1:

```
Dim i As Integer  
i = 1  
Do While i <= 10  
    Cells(i, 1).Value = "test"  
    i = i + 1  
Loop
```

'example 2 (create a row with numbers in your spreadsheet for this demo):

```
Do While ActiveCell.Value <> Empty  
ActiveCell.Value = ActiveCell.Value * 2  
ActiveCell.Offset(0, 1).Select
```

## **Do-Until loop**

'same as the do-while loop, this loop can be used to iterate over a non-numeric condition

'example 1:

```
Dim i As Integer
i = 1
Do until i = 10
    Cells(i, 1).Value = "test"
    i = i + 1
Loop
```

'example 2 (create a row with numbers in your spreadsheet for this demo):

```
Do Until IsEmpty(ActiveCell.Value)
ActiveCell.Value = ActiveCell.Value * 2
ActiveCell.Offset(1, 0).Select
Loop
```

## **Looping through a Collection**

'VBA supports yet another type of looping — looping through each object in a collection of objects. Recall that a 'collection consists of a number of objects of the same type.

'For example, Excel has a collection of all open workbooks (the Workbooks collection), and each workbook has a 'collection of worksheets (the Worksheets collection). When you need to loop through each object in a collection, use 'the For Each-Next structure.

'Example loops through each worksheet in the active workbook and deletes the worksheet if it's empty:

```
Sub DeleteEmptySheets()

    Dim WkSht As Worksheet
    Application.DisplayAlerts = False
    For Each WkSht In ActiveWorkbook.Worksheets
        If WorksheetFunction.CountA(WkSht.Cells) = 0 Then
            WkSht.Delete
        End If
    Next WkSht
    Application.DisplayAlerts = True

End Sub
```

'example 2 (loop to hide all worksheets in the active workbook, except the active sheet).

```
Sub HideSheets()

    Dim Sht As Worksheet
    For Each Sht In ActiveWorkbook.Worksheets
        If Sht.Name <> ActiveSheet.Name Then
            Sht.Visible = xlSheetHidden
        End If
    Next Sht

End Sub
```



'example 3 (unhides all worksheets)

```
Sub UnhideSheets()  
  
Dim Sht As Worksheet  
For Each Sht In ActiveWorkbook.Worksheets  
Sht.Visible = xlSheetVisible  
Next Sht  
  
End Sub
```

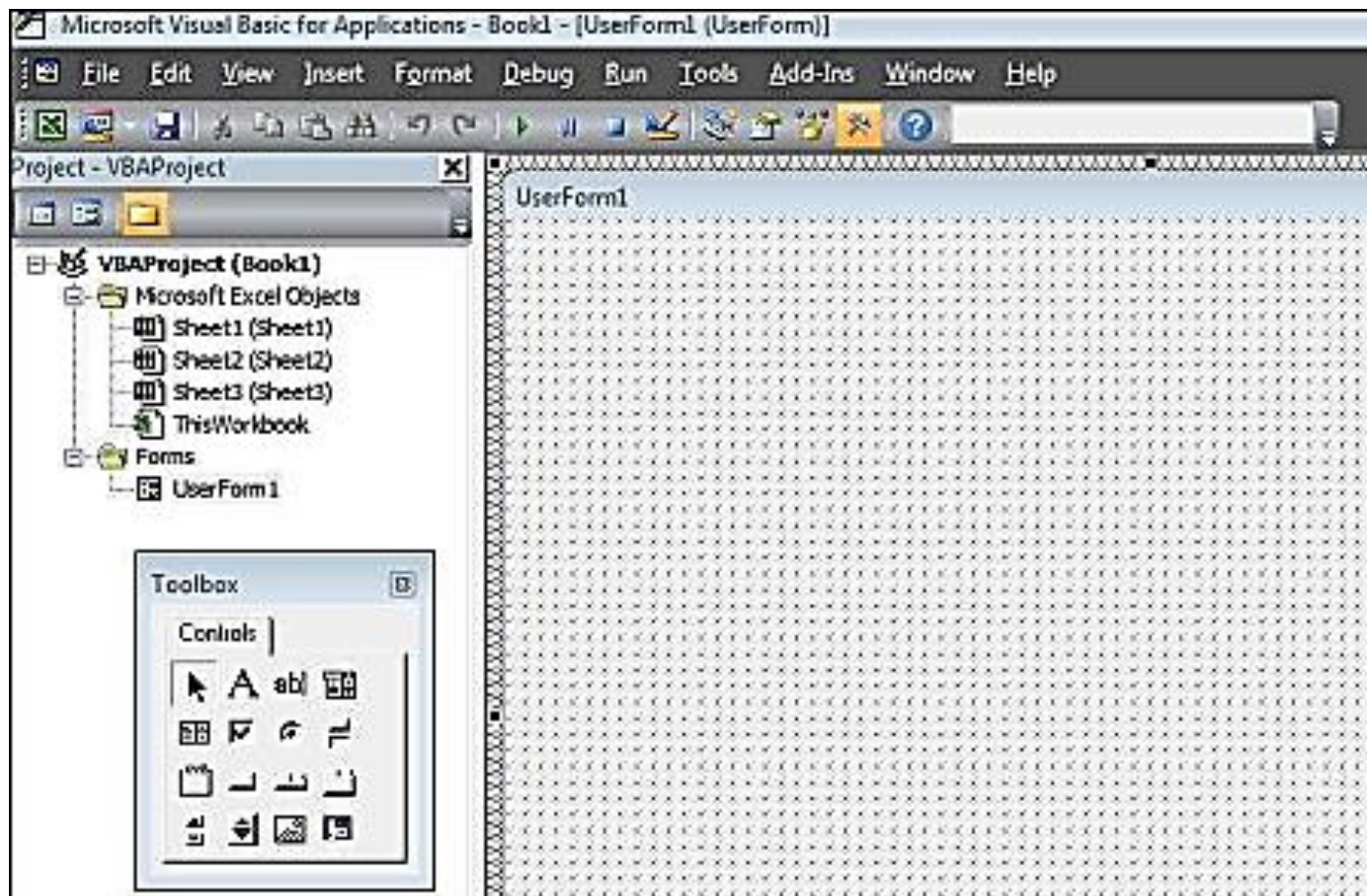
**12nn-1pm**

Lunch Break

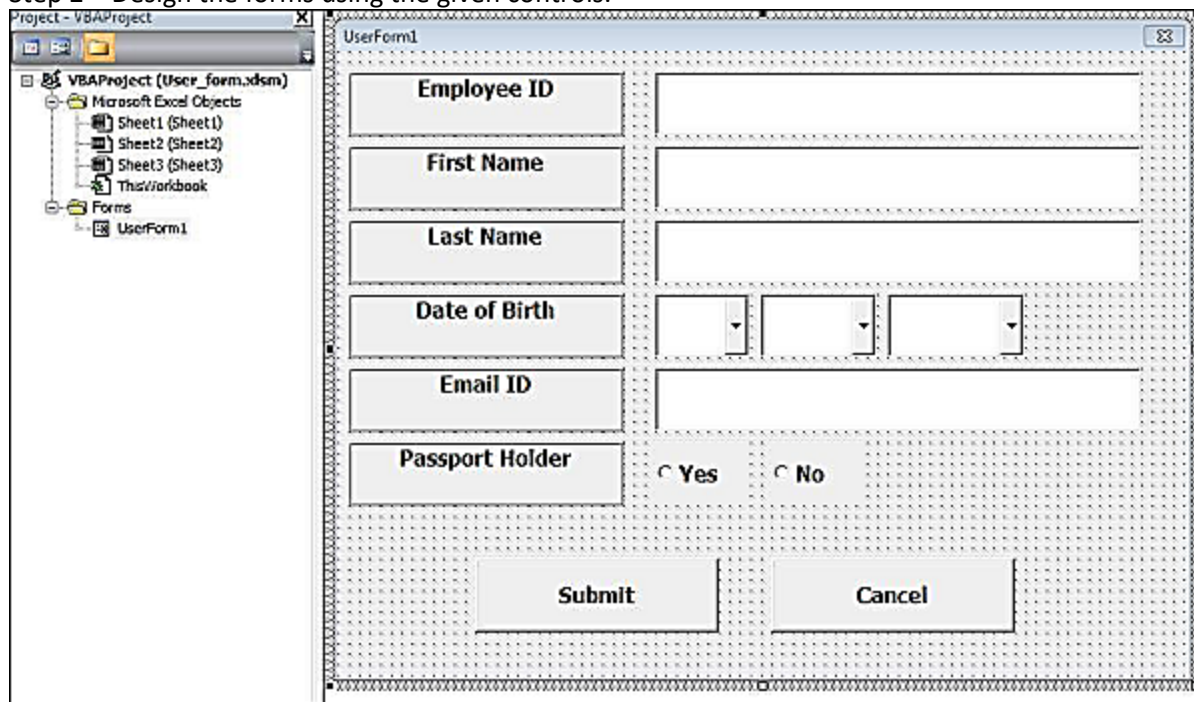
**1pm-2:30pm**

A User Form is a custom-built dialog box that makes a user data entry more controllable and easier to use for the user. In this chapter, you will learn to design a simple form and add data into excel.

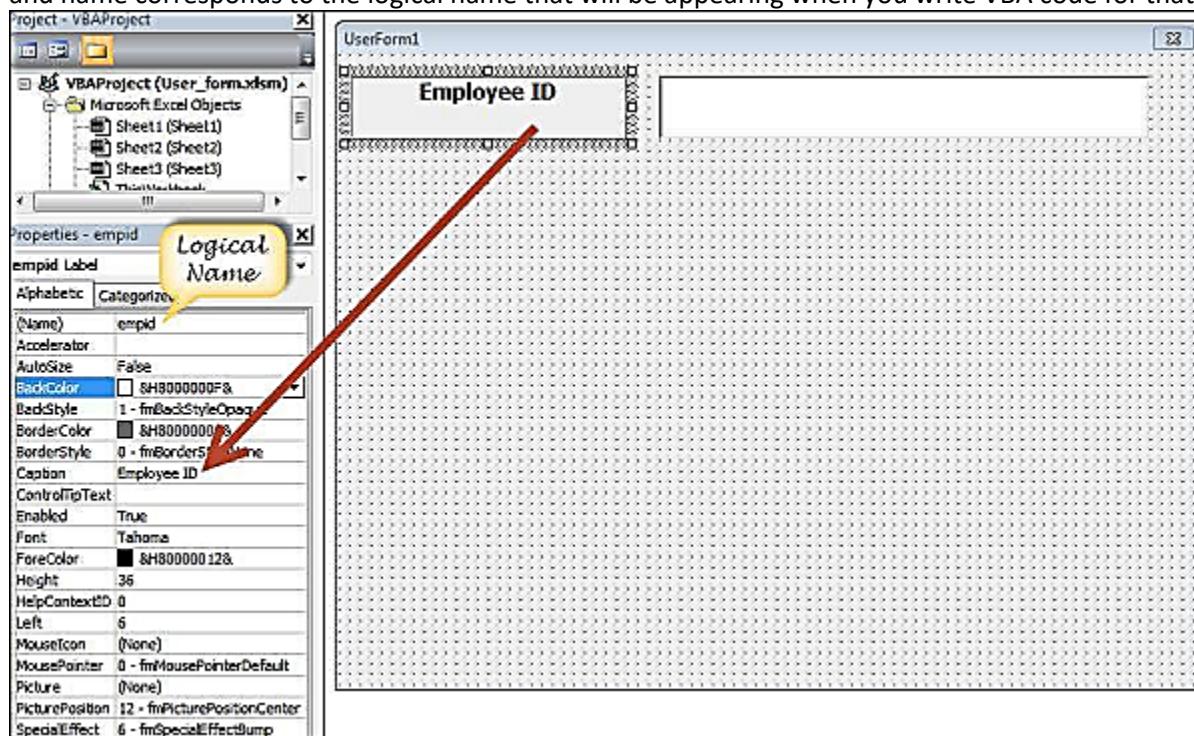
Step 1 – Navigate to VBA Window by pressing Alt+F11 and Navigate to "Insert" Menu and select "User Form". Upon selecting, the user form is displayed as shown in the following screenshot.



Step 2 – Design the forms using the given controls.



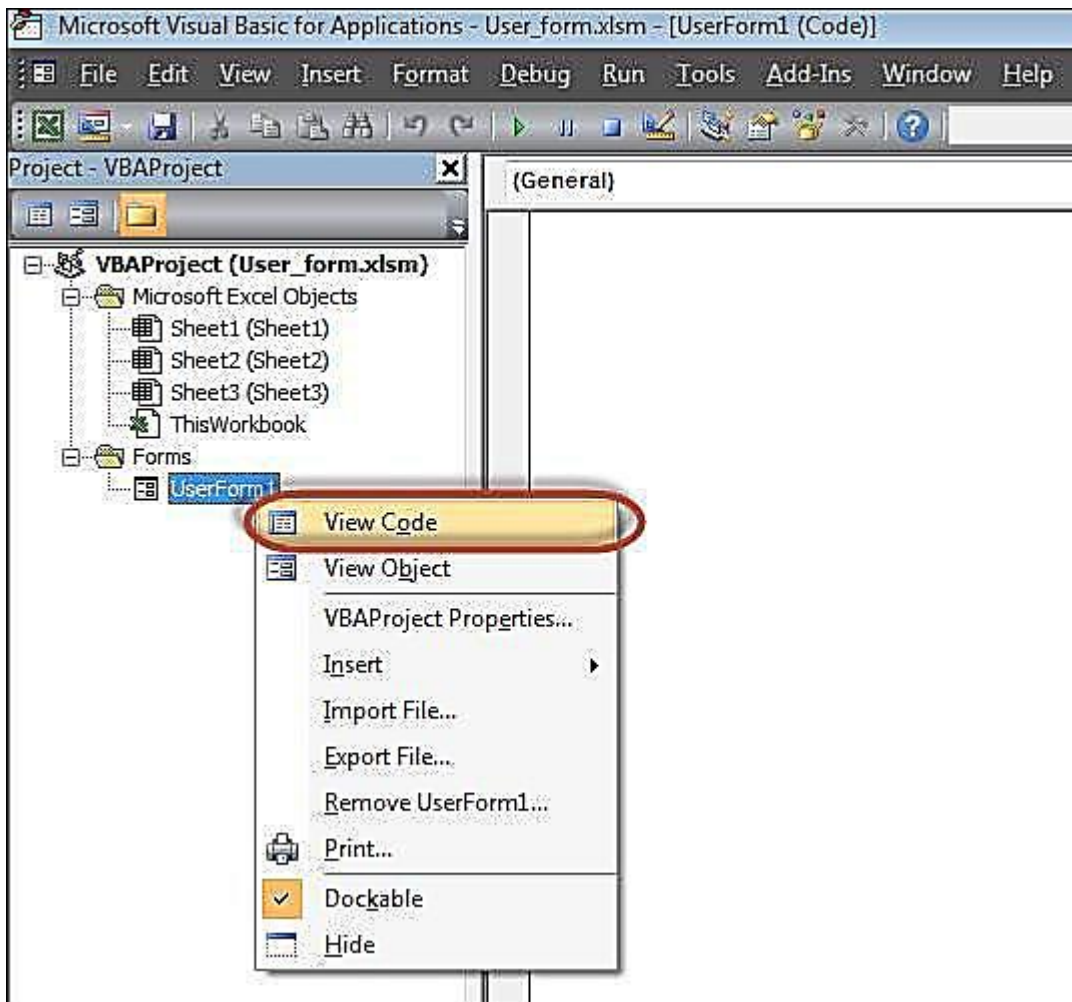
Step 3 – After adding each control, the controls have to be named. Caption corresponds to what appears on the form and name corresponds to the logical name that will be appearing when you write VBA code for that element.



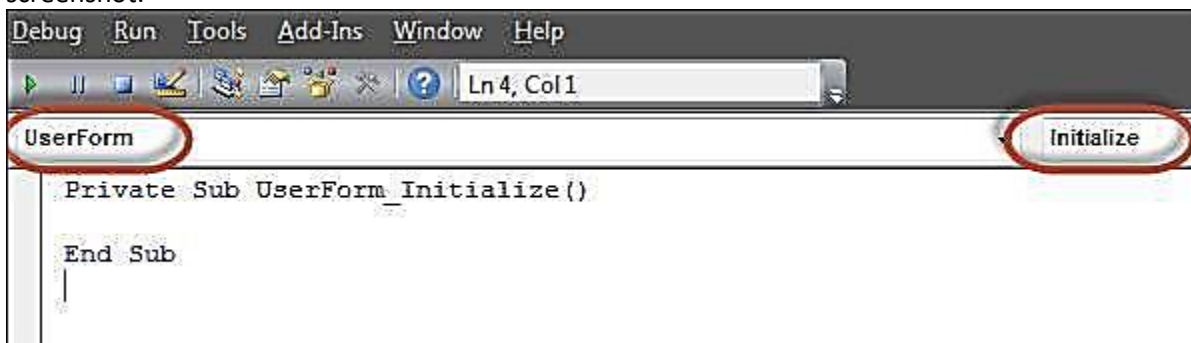
Step 4 – Following are the names against each one of the added controls.

Control	Logical Name	Caption
From	frmempform	Employee Form
Employee ID Label Box	empid	Employee ID
firstname Label Box	firstname	First Name
lastname Label Box	lastname	Last Name
dob Label Box	dob	Date of Birth
mailid Label Box	mailid	Email ID
Passportholder Label Box	Passportholder	Passport Holder
Emp ID Text Box	txtempid	NOT Applicable
First Name Text Box	txtfirstname	NOT Applicable
Last Name Text Box	txtlastname	NOT Applicable
Email ID Text Box	txtemailid	NOT Applicable
Date Combo Box	cmbdate	NOT Applicable
Month Combo Box	cmbmonth	NOT Applicable
Year Combo Box	cmbyear	NOT Applicable
Yes Radio Button	radioyes	Yes
No Radio Button	radiono	No
Submit Button	btnsubmit	Submit
Cancel Button	btncancel	Cancel

Step 5 – Add the code for the form load event by performing a right-click on the form and selecting 'View Code'.



Step 6 – Select 'Userform' from the objects drop-down and select 'Initialize' method as shown in the following screenshot.



Step 7 – Upon Loading the form, ensure that the text boxes are cleared, drop-down boxes are filled and Radio buttons are reset.

```
Private Sub UserForm_Initialize()  
    'Empty Emp ID Text box and Set the Cursor  
    txttempid.Value = ""  
    txttempid.SetFocus  
  
    'Empty all other text box fields  
    txtfirstname.Value = ""  
    txtlastname.Value = ""  
    txtemailid.Value = ""  
  
    'Clear All Date of Birth Related Fields  
    cmbdate.Clear  
    cmbmonth.Clear  
    cmbyear.Clear  
  
    'Fill Date Drop Down box - Takes 1 to 31  
    With cmbdate  
        .AddItem "1"  
        .AddItem "2"  
        .AddItem "3"  
        .AddItem "4"  
        .AddItem "5"  
        .AddItem "6"  
        .AddItem "7"  
        .AddItem "8"  
        .AddItem "9"  
        .AddItem "10"  
        .AddItem "11"  
        .AddItem "12"  
        .AddItem "13"  
        .AddItem "14"  
        .AddItem "15"  
        .AddItem "16"  
        .AddItem "17"  
        .AddItem "18"  
        .AddItem "19"  
        .AddItem "20"  
        .AddItem "21"  
        .AddItem "22"  
        .AddItem "23"  
        .AddItem "24"  
        .AddItem "25"  
        .AddItem "26"  
        .AddItem "27"  
        .AddItem "28"  
        .AddItem "29"  
        .AddItem "30"
```



```
.AddItem "31"
```

```
End With
```

```
'Fill Month Drop Down box - Takes Jan to Dec
```

```
With cmbmonth
```

```
.AddItem "JAN"
```

```
.AddItem "FEB"
```

```
.AddItem "MAR"
```

```
.AddItem "APR"
```

```
.AddItem "MAY"
```

```
.AddItem "JUN"
```

```
.AddItem "JUL"
```

```
.AddItem "AUG"
```

```
.AddItem "SEP"
```

```
.AddItem "OCT"
```

```
.AddItem "NOV"
```

```
.AddItem "DEC"
```

```
End With
```

```
'Fill Year Drop Down box - Takes 1980 to 2014
```

```
With cmbyear
```

```
.AddItem "1980"
```

```
.AddItem "1981"
```

```
.AddItem "1982"
```

```
.AddItem "1983"
```

```
.AddItem "1984"
```

```
.AddItem "1985"
```

```
.AddItem "1986"
```

```
.AddItem "1987"
```

```
.AddItem "1988"
```

```
.AddItem "1989"
```

```
.AddItem "1990"
```

```
.AddItem "1991"
```

```
.AddItem "1992"
```

```
.AddItem "1993"
```

```
.AddItem "1994"
```

```
.AddItem "1995"
```

```
.AddItem "1996"
```

```
.AddItem "1997"
```

```
.AddItem "1998"
```

```
.AddItem "1999"
```

```
.AddItem "2000"
```

```
.AddItem "2001"
```

```
.AddItem "2002"
```

```
.AddItem "2003"
```

```
.AddItem "2004"
```

```
.AddItem "2005"
```

```
.AddItem "2006"
```

```
.AddItem "2007"
```

```
.AddItem "2008"
```

```
.AddItem "2009"  
.AddItem "2010"  
.AddItem "2011"  
.AddItem "2012"  
.AddItem "2013"  
.AddItem "2014"
```

End With

'Reset Radio Button. Set it to False when form loads.

radioyes.Value = False

radiono.Value = False

End Sub

Step 8 – Now add the code to the Submit button. Upon clicking the submit button, the user should be able to add the values into the worksheet.

```
Private Sub btnsubmit_Click()
```

```
    Dim emptyRow As Long
```

```
    'Make Sheet1 active
```

```
    Sheet1.Activate
```

```
    'Determine emptyRow
```

```
    emptyRow = WorksheetFunction.CountA(Range("A:A")) + 1
```

```
    'Transfer information
```

```
    Cells(emptyRow, 1).Value = txttempid.Value
```

```
    Cells(emptyRow, 2).Value = txtfirstname.Value
```

```
    Cells(emptyRow, 3).Value = txtlastname.Value
```

```
    Cells(emptyRow, 4).Value = cmbdate.Value & "/" & cmbmonth.Value & "/" & cmbyear.Value
```

```
    Cells(emptyRow, 5).Value = txtemailid.Value
```

```
    If radioyes.Value = True Then
```

```
        Cells(emptyRow, 6).Value = "Yes"
```

```
    Else
```

```
        Cells(emptyRow, 6).Value = "No"
```

```
    End If
```

```
End Sub
```

Step 9 – Add a method to close the form when the user clicks the Cancel button.

```
Private Sub btncancel_Click()
```

```
    Unload Me
```

```
End Sub
```

Step 10 – Execute the form by clicking the "Run" button. Enter the values into the form and click the 'Submit' button. Automatically the values will flow into the worksheet as shown in the following screenshot.

Employee ID	First Name	Last Name	Date of Birth	Email ID	Passport Holder
100	Jim	Smith	1-Feb-82	Jim@Tutorialspoint.com	Yes
101	Jack	Rose	19-Aug-88	Jack.Rose@Tutorialspoint.com	No
102	Cathy	Nelson	27-Jul-86	cathy@tutorialspoint.com	No

**Employee Form**

**Employee ID**: 102

**First Name**: Cathy

**Last Name**: Nelson

**Date of Birth**: 27 JUL 1986

**Email ID**: cathy@tutorialspoint.com

**Passport Holder**: ☐ Yes ☒ No

**Submit** **Cancel**

**2:30pm-3:30pm**

#### Introduction to Built-In Functions

- ❖ Constants, Expressions and Formulas
- ❖ Fundamentals of Built-In Functions
- ❖ Accessory Built-In Functions

**3:30pm-5pm**

#### Procedures

- ❖ Introduction to Procedures
- ❖ Procedures
- ❖ Introduction to Sub-Procedures
- ❖ Calling a Sub-Procedure
- ❖ Procedures and Access Levels



## Day 3

### 9am-10:30am

#### Introduction to Functions

- ❖ Creating a Function
- ❖ Returning a Value from a Function

### 10:30am-12nn

#### Arguments and Parameters

- ❖ Passing Arguments
- ❖ Techniques of Passing Arguments

### 12nn-1pm

#### Lunch Break

### 1pm-2:30pm

#### Strings

- ❖ Introduction to Strings
- ❖ Introduction to Characters
- ❖ Characters, Strings and Procedures
- ❖ Character and String Conversions
- ❖ The Sub-Strings of a String
- ❖ Other Operations on Strings
- ❖ The Message Box
- ❖ The Input Box

### 2:30pm-4pm

#### Dates and Times in VBA Excel

- ❖ Fundamentals of Dates
- ❖ The Components of a Date
- ❖ Formatting A Date Value
- ❖ Built-In Time Functions
- ❖ The Components of a Time Value
- ❖ Operations on Date and Time Values

### 4pm-5pm

#### Error Handling

- ❖ Handling Errors
- ❖ In Case of Error
- ❖ Types of Error
- ❖ The Err Object