Introduction to VBA for Excel-Tutorial 2

Before we start writing codes, we need to learn the basics. This tutorial will teach you how to name variables and assign data-types in VBA, how to do simple arithmetic operations and hierarchy of operations, how to insert a comment line (very important for documentation), and how to input and output data to and from a VBA code? The answers to these questions are the fundamentals of VBA and soon you will see how they come together to compose codes.

VBA Variable

VBA variable names….

1- Must start with a letter and may be 255 characters long. My advice, don't use long names but use meaningful ones, this will help you in debugging. For example, call a displacement variable ***displac*** instead of just ***x***. Please avoid calling variables ***xx***, ***xxx***, ***xxxx***, etc.

2- Must have no breaks or spaces. For example, ***wave_speed*** so instead of the space between the wave and speed, use underscore as a separator.

3- Not case sensitive that is ***D*** and ***d*** are the same variable. A common mistake is defining the same variable twice with two different values, once lowercase and the other uppercase.

Tidbit: the uppercase letter "**O**" and number "**0**" are hard to differentiate between, similarly lowercase letter "**l**" lowercase and number "**1**". If you want to use "**O**", instead use "**OO**" and for "**l**" use "**ll**" to make them more distinguishable.

To continue on the same subject, let's discuss the variable types. However, VBA does not require variables to be typed (i.e. classified). If you don't declare or define or typed a variable, they are called ***variants***. A list of the most common variable types used in engineering is included below.

Type	Use	Sig. fig.	Storage req.	Definition Method
Integer	Integer	NA	2 bytes	*Dim n as integer*
Single	Decimal	7 figures	4 bytes	*Dim x as single*
Double	Decimal	15 figures	8 bytes	*Dim y as double, z as double*
String*	Text	NA	variable length	*Dim month as string*

* Alternative way to define string date type is by adding $ to the end such that

Dim month$

So to define a variable type you use:

*Dim **variable_name** as **type*** (by changing ***variable_name*** with actual variable name and ***type*** with one of the data types in the table above).

However, VBA does not require declaring data types; it is good programming practice to declare all variables that you are going to use in the code to save memory and shorten the computation time. ***Remember, processor powers are not free.***

Arithmetic Operations

Basic arithmetic operations are addition (+), subtraction (-), multiplication (*), division (/), and raising a number to a power (^). For example,

$a + b$ → addition
$a – b$ → subtraction
$a * b$ → multiplication
a / b → division
$a\string^b$ → raising to power

VBA executes the statements from left to right. You can include more than one statement on the same line such as

$c = a + b$
$d = c\string^2$
or
$c = a + b : d = c\string^2$

Notes:
1- One variable only is allowed on the left of the equals sign. For example, $a + b = c$ is **not permitted**.
2- $c = b + a$, it adds a and b from the memory (previously entered or calculated values). If you perform $c = c + a$, it adds a to previously calculated c then replaces the old value of c with newly calculated value.
3- Sequence of operation: exponentiation is performed before multiplication and division. Multiplication and division are performed before addition and subtraction. VBA perform operations from left-to-right. Parentheses take precedence and are used to group quantities.

VBA Functions

Besides the limited number of functions that are included, we will learn later how to access the more comprehensive Excel function library from VBA. Below see a list of functions that are found in VBA.

Purpose	Function Syntax	Example
Absolute value	Abs(x)	$Abs(-11)=|-11|=11$
arctangent	Atn(x)	$Atn(1.5574)=1$
Cosine	Cos(x)	$Cos(1)=.5403$
Exponential e	Exp(x)	$Exp(3)=e^3=20.086$
Natural log	Log(x)	$Log(3)=1.0986$
Random number	Rnd(x)	
Round to n decimal places	Round(x,n)	$Round(3.141,2)=3.141$
Sine	Sin(x)	$Sin(1)=.8415$
Square root	Sqr(x)	$Sqr(2)=1.414$
Tangent	Tan(x)	$Tan(1)=1.5574$

Tidbit: please be aware that square-root function in Excel is *sqrt(x)*, while in VBA is *sqr(x)*.

Tidbit: engineers use the mathematical constant π all the time, you can code that constant, such that

$$Pi=4*atn(1)$$

Inputs and outputs:

If you remember from the last tutorial, we learned that the basic three components of a program are input, logic and output. Well, it's time to learn how to pass inputs to the code and how to display output. In VBA, such a task can be done in multiple ways.
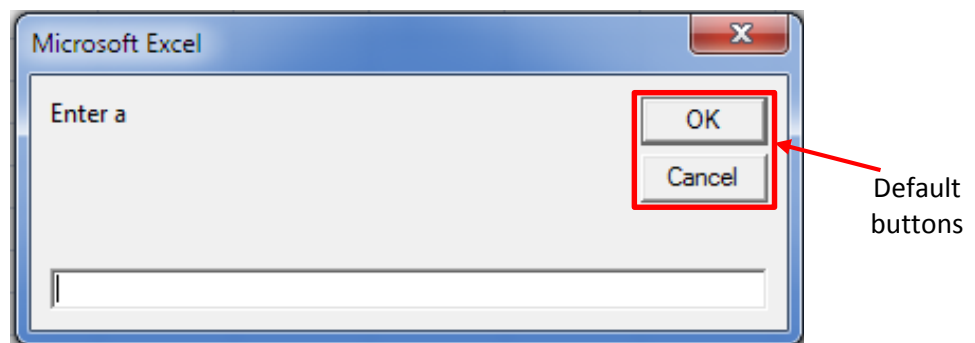
1- Input using *InputBox*

The first way to receive values from the user is **InputBox**, in which you prompt the user to enter one value at a time corresponding to each variable. Suppose, we are composing a code to solve the quadratic equation ($ax^2+bx+c=0$), by the way we will do that later in class, so you want the user to input the coefficients of x^2, x and the constant of the quadratic equation, *a*, *b* and *c*, respectively. Here is how we do it in VBA:

```
Sub inputs_methods()
a = Val(InputBox("Enter a"))
b = Val(InputBox("Enter b"))
c = Val(InputBox("Enter c"))
End Sub
```

The data type of the **InputBox** is string, notice I used the function **Val()** to change the type to double.

When the program is executed, the user will be prompted with:

Note, when you are typing **InputBox** syntax, VBA displays the dynamic help. In other words, VBA is showing you the right syntax.

```
a = InputBox(
    InputBox(Prompt, [Title], [Default], [XPos], [YPos], [HelpFile], [Context]) As String
```

Recall, we enabled this feature during last tutorial. Do you remember how? *Tool/Option/Editor Tab/ Check Auto Quick Info option*.

Tidbit: If you would like to change the title of the ***InputBox***, note the title is "Microsoft Excel" now, the syntax would be:

```
a = Val(InputBox("Enter a", "Example"))
```
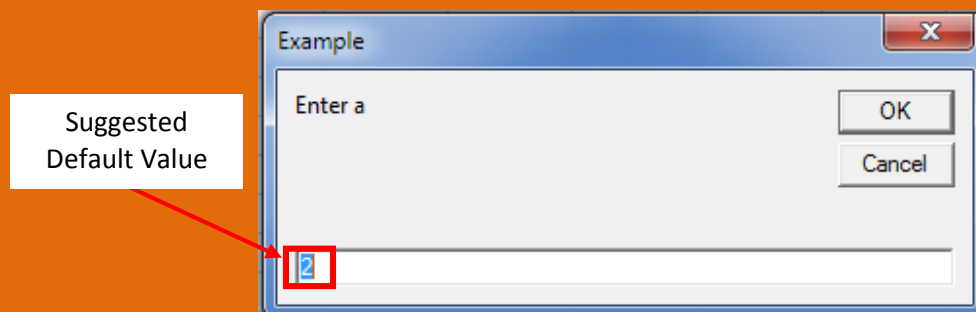
When the program is executed, the user will see:

New Title
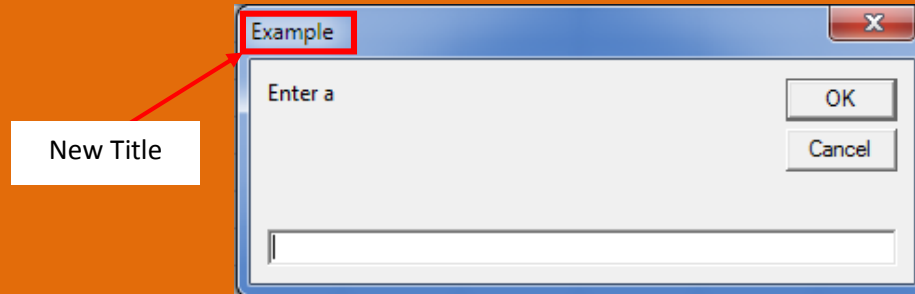
If you would like to suggest a default value to the user, that way the user will have an idea about the range of values, the syntax will be:

```
a = Val(InputBox("Enter a", "Example", 2))
```

When the program is executed, the user will see:

Suggested Default Value

2- Output using *MsgBox* (message box)

Continuing from the first example, that is the quadratic equation example, we will echo back the values to the user to verify that calculation made thereafter is based on the right inputs. Echoing is outputting the values that have been input. The ***MsgBox*** Syntax is:

```
MsgBox "a = " & a & ", " & "b = " & b & ", " & "c = " & c
```

When the program is executed, the user will see:

a = 1, b = 2, c = 3
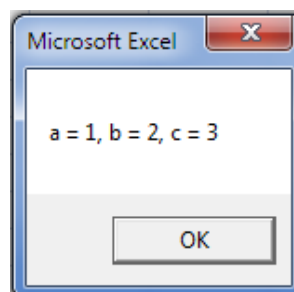
Note that entries of **MsgBox** program lines are separated by ampersand (&) and the text is bounded by the double quotation mark (" "). VBA will output everything inside the quotation mark exactly as entered, so if you need extra spaces or special characters you can add it within the quotation mark.

Tidbit: If you would like to output each coefficient's label and its value on a separate line within the **Msgbox**, use command **Chr(13)** such that:

```
MsgBox "a = " & a & Chr(13) & "b = " & b & Chr(13) & "c = " & c
```

When the program is executed, the user will see:
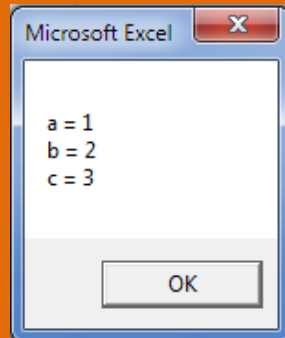
3- Inputs from Worksheet and outputs to Worksheet:

However, the **InputBox** feature is very cool; it is limited to one value at the time. In the example of solving the quadratic equation, it should not be a big issue. However, one can imagine as the engineering problem expands, the number of inputs will dramatically increase and **InputBox** will lose its coolness very quickly. Alternatively, if you have the coefficients stored in worksheet cells, let's say cells A1, A2 and A3 for *a*, *b*, and *c*, respectively. You can access the values as input to your program as follow:

```
a = Cells(1, "A")
```

Or

```
a = Cells(1, 1)
```

The syntax is **Cells(row_index,column_index)**, where *row_index* refers to the row number, and *column_index* is the column number. Note, you can enter the column number either as a letter between double quotations mark or as a number.

One catch about the above commands, it does not specify which worksheet it will grab the values from. Of course, you know that the workbook can have multiple, so which one will be used? Well, it will be whichever the active sheet (i.e. the worksheet that you last worked on). Therefore, we need to be more specific in the following manner:

```
a = Worksheets("Sheet1").Cells(1, 1)
```

Or

```
a = Worksheets(2).Cells(1, 1)
```

To be honest with you, this may be too much work. So let's do it in another way. To make sure that the code accesses the correct worksheet; we will just activate that worksheet on the top of the code. Enter the following command on the top of the code:

```
Worksheets(2).Activate
```

Of course, you must change the sheet index (i.e. "2" between the parentheses) to whichever sheet you stored the data in.

The output is done in the same manner; all you have to do is switch the statements around the equals sign such that:

```
Cells(2, 2) = a
```

Tidbit: let's say you want to output the value of *a* with its units, which is a very good idea, then command would be:

```
Cells(2, 2) = a & " ft/s"
```

When the program is executed, the user will see:

	A	B	C
1			
2		24 ft/s	
3			