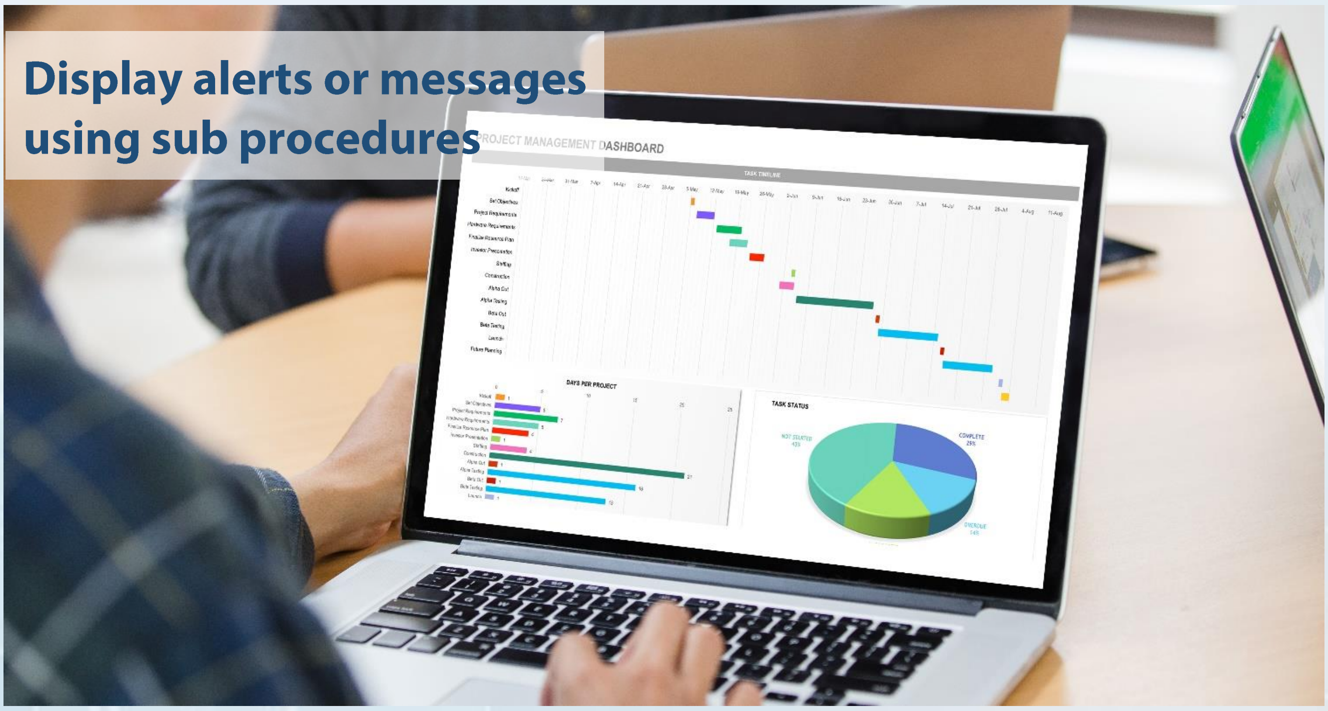
Display alerts or messages using sub procedures



If you need to display messages or alerts using sub procedures, you may use the msgbox object. The syntax for a message box is as follows

```
Msgbox "message"
```

For example, msgbox "Hello!" will give a message box which says Hello! To concatenate a variable you may use "&" symbol.

Example:

```
Sub message()
Name="Pragati"
Msgbox "Hello!" & name
End sub
```

Accept inputs using sub procedures

In the above example, if you need to take the name from the user while the sub procedure is run, you may use **inputbox** object. The syntax of inputbox is as follows

```
Inputbox ("message")
```

Example

Name=Inputbox("Enter the name")

This would accept a name from the user and save it in the variable name. Thus to make the above example even more dynamic, that is to take the name from the user and display a message box with the name, you may use the following code

```
Sub message()
Name=Inputbox("Enter the name")
Msgbox "Hello!" & name
End sub
```

Creating a function

If use a complex formula frequently and wish to have a function that represents the formula so that it becomes easier to use it, you may define the function using the visual basic editor. A function is written in the function block. The syntax of function block is as follows

```
Functionfunction_name(parameter1,parameter2,……)

End function
```

Parameters are inputs to a function. If you want that your function take two inputs, you can have two parameters. These parameters would be variables that would store the values you accept as input. **Note: Variables are memory areas that hold value. Eg: a=2 here a is a variable that holds the value 2** Once defined, you may use a user defined function like you use any worksheet function. In the functions list dialog box, you can find the new function under user defined functions.

Writing a Function Procedure
A Function procedure is a series of Visual Basic statements enclosed by the Function and End Function statements. A Function procedure is similar to a Sub procedure, but a function can also return a value. A Function procedure can take arguments, such as constants, variables, or expressions that are passed to it by a calling procedure. If a Function procedure has no arguments, its Function statement must include an empty set of parentheses. A function returns a value by assigning a value to its name in one or more statements of the procedure.

Syntax of Function procedure

```
[Public | Private | Friend] [Static] Function name [(arglist)] [As type]
[statements]
[name = expression]
[Exit Function]
[statements]
[name = expression]
End Function
```

Example:

```
 Function RectangleArea(SideA As Double, SideB As Double) As Double
   RectangleArea= SideA * SideB
 End Function
```

For example, given below is a sample function to calculate profit.

```
Function Profit (sp,cp)
Profit=sp-cp
End function
```

Variables

Variable is a named storage location that can contain data that can be modified during program execution.
Each variable has a name that uniquely identifies it within its scope. A data type can be specified or not.
Variable names:

- must begin with an alphabetic character,
- must be unique within the same scope,
- can't be longer than 255 characters,
- can't be the same as VBA function or keywords used in VBA,
- can't contain spaces.

Declaring variables

Variables declared implicitly
Variables declared implicitly receive default data type VARIANT

Example:

```
a = 3
```

If you have enabled Option Explicit - implicit declaration of variables is not possible.

Declaration by the Dim statement

```
Dim VariableName As DataType
```

Declaring several variables at once

```
Dim a, b, c as Integer
```

or each variable separately

```
Dim a as Integer

Dim b as Integer

Dim c as Integer
```

Lifetime of variables

The time during which a variable retains its value is known as its lifetime. The value of a variable may change over its lifetime, but it retains some value. When a variable loses scope, it no longer has a value.

a) A procedure-level variable declared with the Dim statement retains a value until the procedure is finished running. If the procedure calls other procedures, the variable retains its value while those procedures are running as well.

b) In a standard module or a class module, it retains its value until you stop running your code.

Operator Precedence

When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence.

When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. Comparison operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear. Arithmetic and logical operators are evaluated in the following order of precedence:

Arithmetic Operators	**Comparision Operators**	**Logical Operators**
raise a number to the power (^)	equal (=)	Not
negative value (-)	difference(<>)	And
multiply, divide(*, /)	less than(<)	Or
dividing (\)	greater than (>)	Xor
modulo (**Mod**)	less than or equal to (<=)	Eqv
sum, subtract(+, –)	greater than or equal to (>=)	Imp
concatenation (&)	Like	
	Is	

Using conditions in a function

To use conditions in a function, you may want to do a calculation where, tax is calculated as follows,
Tax is 5% of salary if salary is less than 5000, 7% if salary is between 5000 and 10000 and for all the others it is 10%. To do this in a function, use the "If---Else--Endif" block. The syntax for the same is as follows
Using the discussed concept, the scenario discussed earlier can be solved as below

```
Function Ptax(sal)
If sal< 5000 Then
Ptax = sal * .05
ElseIfsal> 5000 And sal<= 10000
Then

Ptax = sal * .07
Else: Ptax = sal * .1
End If
End Function
```

```
If condition then
Statement
Else
Statement
Endif
```

Here, else is an optional syntax. There may exist an If-Endif block without an else. If there are multiple conditions, you may use Elseif instead of else. The syntax of if with Elseif is as follows,

```
If condition then
Statement
Elseif
Statement
Endif
```

Select Case Statement

```
Select Case testExpression
  [Case expressionList-n
    [statements-n]] ...
  [Case Else
    [elseStatements]]

 End Select
```

Loops

Looping allows you to run a group of statements repeatedly. Some loops repeat statements until a condition is False; others repeat statements until a condition is True. There are also loops that repeat statements a specific number of times or for each object in a collection.

For ... Next

You can use For...Next statements to repeat a block of statements a specific number of times. For loops use a counter variable whose value is increased or decreased with each repetition of the loop.

```
For counter = start To end [Step step]
```

```
  [statements]
  [Exit For]
  [statements]
Next [counter]
```

example:

```
Sub ForLoop_example()
  Dim i As Long
  For i = 1 to 5
    MsgBox i
  Next
End Sub
```

For Each ... Next
For Each...Next statements repeat a block of statements for each object in a collection or each element in an array. Visual Basic automatically sets a variable each time the loop runs.

```
Dim variableName As Variant
For Each variableName In collectionName
  [statements]
  [Exit For]
  [statements]
Next variableName
```

example:

```
Sub ForEachLoop_example()
  Dim wks As Variant
  For Each wks In Worksheets
    MsgBox wks.Name
  Next arkusz
End Sub
```

Do While (or Until) ... Loop
You can use Do...Loop statements to run a block of statements an indefinite number of times. The statements are repeated either while a condition is True or until a condition becomes True.

Repeating Statements While a Condition is True
Repeating Statements Until a Condition Becomes True

There are two ways to use the While (and Until) keyword to check a condition in a Do...Loop statement. You can check the condition before you enter the loop, or you can check it after the loop has run at least once.

```
Do [{While | Until} condition]
  [statements]
  [Exit Do]
  [statements]
Loop
```

```
Do
  [statements]
  [Exit Do]
  [statements]
Loop [{While | Until} condition]
```

example:

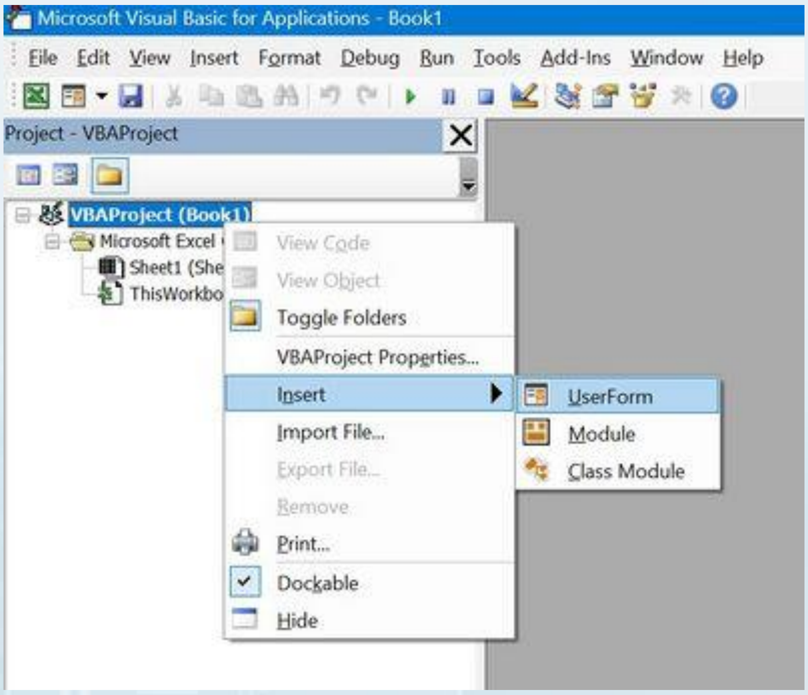
```
Sub DoUntil_example()
  Dim x
  Do
    x = InputBox("")
    Debug.Print x
  Loop Until x = ""
End Sub
```

Exit For/Do

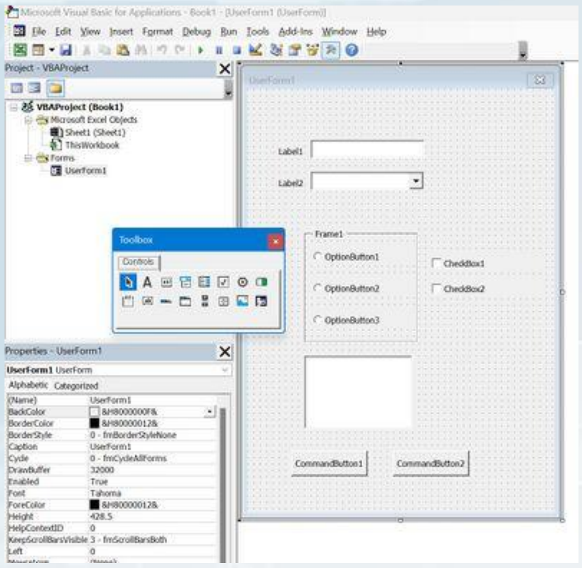
You can exit a For...Next statement before the counter reaches its end value by using the Exit For statement. The same applies for the For Each ... Next statement and Do ... Loop. In the last case use Exit Do.

Userforms

In VBA Excel, user forms are graphical interfaces that allow users to interact with the data and functionality of the worksheet. They are created using the UserForm object, which provides a container for various controls such as buttons, text boxes, list boxes, combo boxes, and others.
To create a user form in VBA Excel, follow these steps:
Open the VBA Editor by pressing Alt + F11 or by selecting Developer -> Visual Basic from the Excel ribbon.
Right-click on the project or workbook where you want to create the user form, and select Insert -> UserForm.



This will create a blank user form, where you can add various controls from the Toolbox. To open the Toolbox, select View -> Toolbox from the menu.



(Name)	UserForm1
BackColor	&H8000000F&
BorderColor	&H80000012&
BorderStyle	0 - fmBorderStyleNone
Caption	UserForm1
Cycle	0 - fmCycleAllForms
DrawBuffer	32000
Enabled	True
Font	Tahoma
ForeColor	&H80000012&
Height	428.5
HelpContextID	0
KeepScrollBarsVisible	3 - fmScrollBarsBoth
Left	0

Drag and drop the controls you want to use onto the user form. You can resize and position them as needed.
To add functionality to the controls, you can write VBA code in the corresponding event procedures. For example, you can write code in the Click event of a button to perform a specific action when the user clicks on it.

```
Private Sub UserForm_Initialize()

  With ComboBox1
    .Clear
    .AddItem "New York"
    .AddItem "Paris"
    .AddItem "London"
    .AddItem "Berlin"
    .AddItem "Warsaw"
    .AddItem "Vienna"
  End With

  TextBox1 = ""

  OptionButton1 = True
  CheckBox1 = False
End Sub

Private Sub CommandButton1_Click()
  Unload Me
End Sub

Private Sub CommandButton2_Click()
  Call UserForm_Initialize
End Sub
```

Once you have finished designing and coding the user form, you can display it by calling its Show method from a VBA procedure or event.

Naming Convention

Using a naming convention for form controls in VBA has several benefits:

- Readability and clarity: Naming convention makes the code more readable and easier to understand. By using meaningful and consistent names for the form controls, it becomes easier to identify what each control does and how it's used.
- Consistency: When using a naming convention, it ensures that all the controls have a consistent and predictable naming pattern, which makes the code more organized and easier to maintain.
- Avoiding naming conflicts: By using a naming convention, it helps to avoid naming conflicts between different controls. This is especially important when working on larger projects with many forms and controls.
- Compatibility with other developers: Using a naming convention is also beneficial when working with other developers, as it ensures that everyone follows the same naming convention, making the code more cohesive and easier to collaborate on.

Overall, using a naming convention for form controls in VBA can make the code more readable, organized, and easier to maintain, which can save time and reduce errors in the long run.

The naming convention for form controls:

1. Text box: txt (e.g. txtFirstName)
2. Label: lbl (e.g. lblLastName)
3. Command button: cmd (e.g. cmdSave)
4. Check box: chk (e.g. chkAgree)
5. Option button: opt (e.g. optMale)
6. List box: lst (e.g. lstColors)
7. Combo box: cbo (e.g. cboCountry)
8. Image: img (e.g. imgLogo)
9. Frame: fra (e.g. fraAddress)
10. Scroll bar: sbr (e.g. sbrZoom)

Again, the specific naming convention chosen may depend on personal preference and project standards, but using a consistent naming convention can make the code more organized and easier to read and maintain.