

# IFA/QFN VBA Tutorial

Notes prepared by Keith Wong

## Chapter 2: Basic Visual Basic programming

### 2-1: What is Visual Basic

BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code. It is a type of programming language. Visual Basic is constructed based on BASIC.

Visual Basic differs from BASIC in mainly two ways.

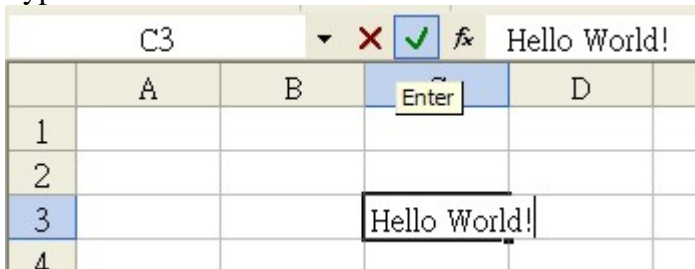
1. Difference in syntax for a lot of commands. Also there are a lot more commands.
2. Programming “visually”: you can create programs by mouse clicking and see some of your results before running your program.

Therefore you may treat Visual Basic as a *quite* different language from BASIC.

### 2-2: The First “Hello World” Program

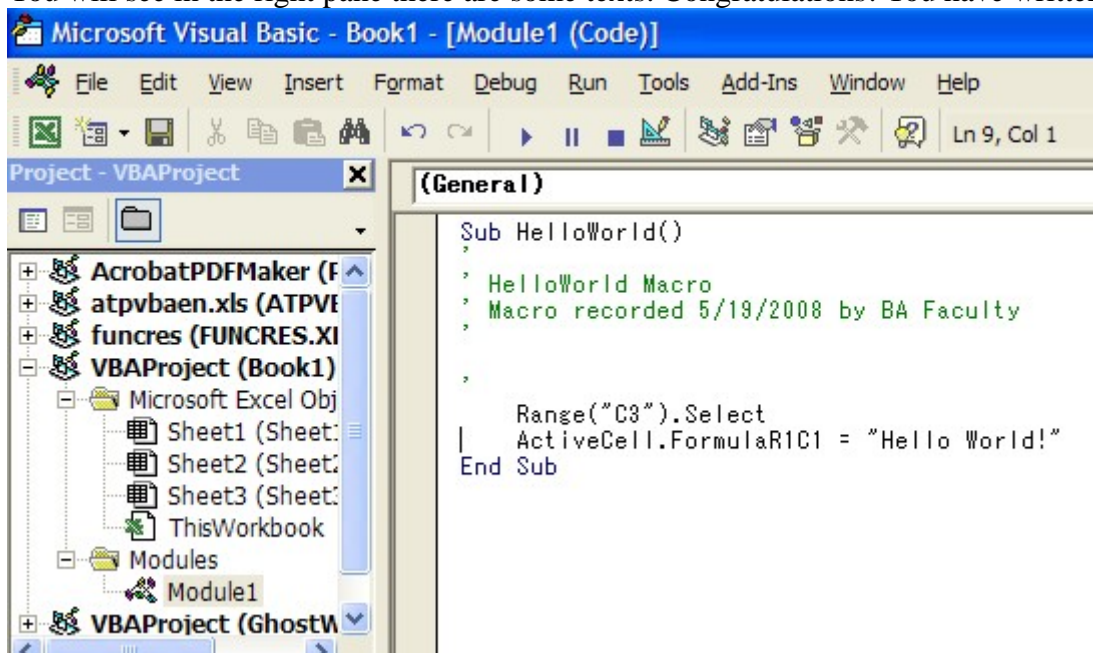
#### Let's try! Corner

- 1 In Excel, click Tools > Macro > Record New Macro...
- 2 Type “HelloWorld” in the Macro name textbox, and then click OK.
- 3 Select cell C3
- 4 Type “Hello World!”. Press the “tick” button next to the formula bar.



- 5 Click the stop recording macro button.
- 6 Click Tools > Macro > Visual Basic Editor.
- 7 In the upper left pane, open VBAProject (Book1) > Modules > Module1.

You will see in the right pane there are some texts. Congratulations! You have written your first program.



We will study the structure of this program.

```
Sub HelloWorld()
```

- This is the beginning of the procedure (or treat it as a program at this point) “HelloWorld”.

```
' HelloWorld Macro
```

```
' Macro recorded <date> by <user>
```

- These two lines (or any lines beginning with ') are comments. Comments are not processed by the computer. So you can add anything you wish after '. For example, you can add reminders or explanations for the codes. Another usage is “commenting out” a line of code. This is very useful in debugging, when you want to test the program without a specific line.

```
Range(“C3”).Select
```

- Range(“C3”) is to specify the cell range “C3”. You can specify more than one cell, e.g. “A1:B10”.
- .Select is to select the cell range specified.

```
ActivateCell.FormulaR1C1 = “Hello World!”
```

- ActivateCell is your current cell selected.
- .FormulaR1C1 is the formula of your cell in “R1C1” format. (“R1C1” format is a method of referring a cell. You will learn this later.)
- “Hello World!” is a string. A string is a sequence of characters. You need “”(double quotation marks) to enclose a string in writing a program.

```
End Sub
```

- This is the end of the procedure “HelloWorld”.

Normally you write codes in procedures. With each procedure there are a beginning line and an ending line to tell the computer where the beginning and the end of the procedure are.

### 2-3: Variables

Variables are storage of certain values or objects. Every variable has a type, which means what type of values it is storing.

In Visual Basic, you need not declare every variable you use. If you want to make this mandatory, you can add a statement “Option Explicit”. This can protect you against typographical errors.

To declare a variable, use the following statement:

```
Dim varname As vartype
```

where varname is the variable name, and vartype is the variable type.

To declare several variables, you can put several Dim statements together like this:

```
Dim count1 As Integer, count2 As Integer, salesAmount As Double
```

Note that if you declare Dim statements like this you will not get what you may be expected:

```
Dim count1, count2, count3 As Integer
```

count1 and count2 will be declared as a Variant type.

A variable name must begin with a letter, following with alphanumeric characters or underscores (\_). “Alphanumeric characters” means “numeric and letter characters”. The total length of the variable name should be less than 256 characters, and should not be a VBA keyword, such as Sub, Function, End, For, Optional, New, Next, Nothing, CStr, Integer, String.

You should name your variables, functions and other objects with a meaningful name to increase readability. Also, comments can be added next to the variables to describe them.

Variables can be classified into three basic types: Numeric data type, String data type, Date data type, and Variant data type.

### 2-3-1: Numeric Data Type

Type	Short hand	Range	Description
Byte		0 to 255	Unsigned, integer number
Currency	@	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	A signed, fixed-point number with up to 15 digits to the left of the decimal and 4 digits to the right
Decimal		+/-79,228,192,514,264,337,593,543,950,335 with no decimal point and +/-7.9228162514264337593543950335 with 28 digits behind the decimal point.	Cannot be directly declared in VBA; requires the use of a Variant data type
Double	#	+/-1.79769313486231E308 to +/-4.94065645841247E-324	Signed double-precision floating-point number
Integer	%	-32,768 to 32,767	Signed integer number
Long	&	-2,147,483,648 to 2,147,483,647	Signed integer number
Single	!	+/-3.402823E38 to +/-1.401298E-45	Signed single-precision floating-point number
Boolean		True(-1) or False(0)	Truth value

You should choose your data type according to your needs. For example if you expect your variable holds only values from 1 to 100, use Integer instead of Long or Double since those data types require more storage in the computer, and it can slow your computer if you are running simulations, or anything requiring a lot of computations.

Regarding the short-hands, you can declare your variables, for example, an integer with name count by the following method:

```
Dim count As Integer
Dim count%
```

One remark here is that although there are shorthand symbols for some of the data types, I do not recommend you to use those symbols as this will reduce the readability of the code.

### 2-3-2: String Data Type

“String” is a sequence of characters. You can declare a string variable like this:

```
Dim stuName As String
Dim stuChiName$
```

stuName will be of a variable-length String. If you want a fixed-length String, you can declare like this:

```
Dim stuID As String * 8
```

This will be a fixed-length string with length 8.

### 2-3-3 Date Data Type

“Date” data type can hold dates and times separately and simultaneously. You can declare a date variable as follows:

```
Dim startDate As Date
```

### 2-3-4: Variant Data Type

VBA supports variant data type, which means your variable can hold almost any type of data. VBA will try to convert the variable to the data type which can hold the input data. You can declare a variant variable with type set to *Variant*, or you can omit the *As vartype* part.

```
Dim c1 As Variant
Dim c2
```

Apart from normal data, variant type variable can also hold three special types of value: error code, *Empty* (it indicates that the variable is empty, and is not equal to 0, *False*, an empty string or other values) and *Null* (it means that the variable has not been assigned memory, and is not equal to 0, *False*, an empty string, *Empty*, or other values).

### 2-4: Assignment Statements

We can assign values to variables with the assignment operator (=). For example,

```
stuName = "Chan Tai Man"
stuID = "07000001"
count1 = 1
count2 = 30000&
startDate = #1/1/2008#
startDateTime = #1/1/2008 23:01:20#
salesAmount = 2423500#
todayDate = CDate("31st May, 2008")
```

Each value also has a type associated to it.

Type	Symbol	Data Type Conversion Function	Example	Remarks
Byte		CByte( <i>expr</i> )	CByte(10)	
Currency	@	CCur( <i>expr</i> )	2345.67@, CCur(3.4567E7)	Rounding to four decimal places.
Decimal		CDec( <i>expr</i> )	CDec(3.4567)	Rounding to a variable number of decimal places dependent upon size of number. Returns a variant typed expression.
Double	#	CDbl( <i>expr</i> )	1.234, 2.3456#, CDbl(3.46567E-10)	
Integer	%	CInt( <i>expr</i> )	200, 300%, CInt(456.78)	.5 will round to the nearest even integer.
Long	&	CLng( <i>expr</i> )	100&, 3E6&, 456789, CLng(3445676.234)	.5 will round to the nearest even integer.
Single	!	CSng( <i>expr</i> )	2.345!, CSng(2.34556674)	
Boolean		CBool( <i>expr</i> )	True, False, CBool(10)	Any non-zero expression will result in True (-1).
String	“ ”	CStr	“String”, “”, CStr(1.2345)	
Date	##	CDate( <i>expr</i> )	#1/1/1984#, #15 July 1999#, #8:47 PM#, #1982/05/07 15:45:23#, CDate(30000)	Numeric expressions will return a date matching number of days from January 1, 100. String expressions will return an interpreted date.

If the value type does not equal to the variable type in assignment statements, VBA will try to convert them automatically. This can lead to loss of data. For example, if you assign a double value to an integer variable, the value will be rounded to the nearest even integer.

### 2-5: Constants

You define constants with the keyword **Const**. Constants are storages whose values will not change. For example, you can define constants as follows.

```
Const maxSize = 3
Const piOver180 As Double = 3.1415926535 / 180
```

Also VBA has defined a lot of intrinsic constants for use with procedures and functions.

## 2-6: Operators

The operator description and precedence is as follows (with a higher precedence at the top of the list).

- Exponentiation (^)
- Unary negation (-)
- Multiplication and division (\*, /)
- Integer division (\)
- Modulus arithmetic (Mod)
- Addition and subtraction (+, -), string concatenation (+)
- String concatenation (&)
- All comparison operators (=, <>, <, <=, >, >=, Like, Is, TypeOf...Is)
- Not
- And
- Or
- Exclusive Or(Xor) (returns true if one of *a* and *b* is true, but not both)
- Equivalence(Eqv) (returns true if *a* and *b* are both true or both false)
- Implication(Imp) (return true if *a* is false or *a* and *b* are both true)

Operator precedence means the priority that VBA interprets a statement. For example,

$3 + 4 * 5 / 6$   
will be interpreted as  
 $3 + ((4 * 5) / 6)$

The expression

"My " & "house " + "has fallen"  
will be interpreted as  
"My " & ("house " + "has fallen")

As demonstrated above, you can increase the precedence of an operator by including parentheses (()).

Before we practice some simple programming technique, we should introduce the message box for output and the input box for input.

## 2-7: Message Box

The syntax for a message box function is:

- `MsgBox(prompt[, buttons][, title][, helpfile, context]) as Integer`
- **prompt:** Required string parameter. The text to be displayed in the message box. The text can hold around 1024 characters. If you want to have multiple lines, you can include `vbCrLf` to specify a new line.
  - **buttons:** Optional numeric parameter. A intrinsic constant can be inputted to show different styles of buttons, icons and styles in the message box. See below for a complete list of options available. You can use "+" operator to to specify a button and an icon simultaneously.
  - **title:** Optional string parameter. Specifies the title of the message box. If omitted, "Microsoft Office Excel" is displayed.
  - (Advanced) **helpfile:** Optional string parameter. Specifies a help file used.
  - (Advanced) **context:** Optional numerical parameter. Specifies the Help context number.
  - **Return value:** An integer to indicate which button was clicked by the user. See below for a list of the values.

Button Options for Message Boxes:

<b>Intrinsic Constant</b>	<b>Value</b>	<b>Description</b>
<b>Buttons</b>		
vbOkOnly	0	Displays an OK button only.
vbOkCancel	1	Displays an OK and a Cancel button.
vbAbortRetryIgnore	2	Displays three buttons labeled Abort, Retry, and Ignore.
vbYesNoCancel	3	Displays three buttons labeled Yes, No, and Cancel.
vbYesNo	4	Displays buttons labeled Yes and No.
vbRetryCancel	5	Displays buttons labeled Retry and Cancel.
<b>Icons</b>		
vbCritical	16	Displays a solid red circle enclosing a white <i>X</i> .
vbQuestion	32	Displays a cartoon balloon enclosing a question mark.
vbExclamation	48	Displays a yellow triangle enclosing an exclamation point.
vbInformation	64	Displays a cartoon balloon enclosing a lowercase letter <i>i</i> .
<b>Default Button</b>		
vbDefaultButton1	0	Sets the first button as the default button.
vbDefaultButton2	256	Sets the second button as the default button.
vbDefaultButton3	512	Sets the third button as the default button.
vbDefaultButton4	768	Sets the fourth button as the default button.
<b>Modality</b>		
vbApplicationModal	0	Marks the message box as application modal. Stops all processing of the current application until the message box is dismissed. Does not interfere with any other applications.
vbSystemModal	4096	Marks the message box as system modal; it will always appear as the topmost window regardless of which application the user switches to.
vbMsgBoxHelpButton	16384	Adds a Help button to the message box.
vbMsgBoxSetForeground	65536	Causes the message box to be displayed in the foreground.
vbMsgBoxRight	524288	Causes the text in the message box to be right-aligned.
vbMsgBoxRtlReading	1048576	Causes the text to be displayed right-to-left on Hebrew and Arabic systems.

Values Returned by a Message Box:

<b>Intrinsic Constant</b>	<b>Value</b>	<b>Description</b>
vbOk	1	The OK button is pressed.
vbCancel	2	The Cancel button is pressed.
vbAbort	3	The Abort button is pressed.
vbRetry	4	The Retry button is pressed.
vbIgnore	5	The Ignore button is pressed.
vbYes	6	The Yes button is pressed.
vbNo	7	The No button is pressed.

Some examples:

```
MsgBox "A Test"
```

- Displays “A Test” in a pop-up message box with an OK button.

```
result = MsgBox("A Test?", vbYesNo + vbQuestion, "Testing")
```

- Displays “A Test?” in a pop-up message box, with “Testing” as heading, a Yes button and a No button, and a question mark icon. A numeric value of `vbYes` or `vbNo` will be assigned to the variable `result`, depending on which button the user clicks.

```
result = MsgBox("Error. Retry?", vbRetryCancel + vbCritical + vbSystemModal, "Error")
```

- Displays “Error. Retry?” in a pop-up message box with a retry button and a cancel button, “Error” as title, and an “X” icon. When you click on other applications, the message box stays in foreground. A numeric value of `vbRetry` or `vbCancel` will be assigned to the variable `result`, depending on which button the user clicks.

## **2-8: Input Box**

The syntax for an input box function is:

```
InputBox(prompt[, title][, default][, xpos][, ypos][, helpfile, context] as String
```

- **prompt**: Required string parameter. The text to be displayed with the input box. The text can hold around 1024 characters. If you want to have multiple lines, you can include a `vbCrLf` to specify a new line.
- **title**: Optional string parameter. Specifies the title of the message box. If omitted, “Microsoft Office Excel” is displayed.
- **default**: Optional string parameter. The text specified will be put in the input box as a default input.
- **xpos**: Optional numeric expression. The number of twips from the left edge of the screen to left edge of the input box. A trip (twentieth of a point) is  $1/1440^{\text{th}}$  of an inch or  $1/567^{\text{th}}$  of a centimeter. If omitted, the input box is centered horizontally.
- **ypos**: Optional numeric expression. The number of twips from the top edge of the screen to top edge of the input box. If omitted, the input box is centered vertically.
- (Advanced) **helpfile**: Optional string parameter. Specifies a help file used.
- (Advanced) **context**: Optional numerical parameter. Specifies the Help context number.
- **Return value**: A string which returns the inputted value. If the text box is blank or the Cancel button is pressed, the string will be empty.

Some examples:

```
a = InputBox("Input a")
```

- Displays an input box with text “Input a”. The input is assigned to the variable `a`.

```
b = InputBox("Input b", "Test", "10")
```

- Displays an input box with text “Input b” with heading “Test”. The default value in the input box is “10”. The input is assigned to the variable `b`.

```
c = InputBox("Input c", "Test", "10", 1440, 1440)
```

- Displays an input box with text “Input c” with heading “Test”. The default value in the input box is “10”. The box is displayed at 1 inch away from each of the left and top edges of the screen. The input is assigned to the variable `c`.

### Let's try! Corner

We try to find the present value of an  $n$ -year annuity immediate with annual payment of  $R$  and effective interest rate  $i$ . The user will input  $R$ ,  $n$ ,  $i$  from three prompts and the output will be produced in a message box. The formula for the present value is

$$Ra_{\overline{n}|i} = R \cdot \frac{1 - (1 + i)^{-n}}{i}$$

- 1 Open Excel.
- 2 Set up cells with initial values  $R = 100$ ,  $n = 10$ ,  $i = 5\%$  like the figure below.

	A	B
1	R	100
2	n	10
3	i	5%
4	PV	
5		

- 3 Record the following macro: In cell B4, enter the formula for the present value.
- 4 Check if the PV is correct (772.17). If not, record the macro again.
- 5 Open Visual Basic Editor. Open "Module1".
- 6 Check that you see the subroutine of the newly created macro.
- 7 You will see something like this:  
`ActiveCell.FormulaR1C1 = "=R[-3]C*(1-(1+R[-1]C)^-R[-2]C)/R[-1]C"`  
All other codes should be rubbish and you can delete them except this one.
- 8 Declare three variables for the inputs and one variable for the output, above the statement in step 7.  
`Dim R As Currency, n As Integer, i As Double, PV As Currency`
- 9 Add 3 InputBox functions under the declaration statement for the inputs.  
`R = InputBox("Input R")  
n = InputBox("Input n")  
i = InputBox("input i")`
- 10 Modify the statement in step 7. Replace "`R[*]C`" with the correct variables. Also remove the double quotes. Change the left part of the assignment statement to PV. You will get  
`PV = R * (1 - (1 + i) ^ -n) / i`
- 11 Add a message box for output.  
`MsgBox PV`
- 12 Test the subroutine by running the macro, inputting  $R = 100$ ,  $n = 10$ ,  $i = 0.05$ .

### Let's try! Corner

We try to calculate the forward price of a stock by repeating the above process. The forward price of a stock is given by

$$F_{0,T}(S) = S_0 e^{(r-\delta)T}$$

where  $S_0$  is the initial stock price,  $r$  is the continuous interest rate,  $\delta$  is the continuous dividend rate, and  $T$  is the time to delivery.

## 2-9: Comments and Codes Continuing on Next Line

You can add a comment to the code by preceding the comments with ' or using the keyword Rem at the beginning of a line.

You can continue a statement at the next line, if you find it too long, with the symbol \_(underscore). For example, this is counted as one statement

```
Dim R As Currency, n As Integer, _  
i As Double, PV As Currency
```

## 2-10: Conditional Statements

When you want to make a decision within the code about user inputs or any values that might be calculated, you can use conditional statements.

### 2-10-1: If...Then...Else Statement

There are two forms of the If...Then...Else statement: either single-line or multi-line. The syntax is:

```
If condition Then statement [Else elseStatement]
```

or

```
If condition Then  
  [statements]  
[ElseIf elseifcondition Then  
  [elseifStatements]]  
[ElseIf elseifcondition2 Then  
  [elseifStatements2]]  
...  
[Else  
  [elseStatements]]  
End If
```

You can only insert one statement in the single-line form, but many statements in the multi-line form.

For the condition part, you need to specify an expression which can evaluate to True or False. The common comparison operators are =, <>, <, <=, >, >=. (<> means "not equal", others are self-explanatory) Apart from these, you can also use logical operators Not, And, Or, Xor (Exclusive Or), Eqv (Equivalence) and Imp (Implication). The truth tables are shown below:

	<i>a</i>	
	<i>True</i>	<i>False</i>
<b>Not a</b>	False	True

<b>a And b</b>	<i>a</i>	
<i>b</i>	<i>True</i>	<i>False</i>
<i>True</i>	True	False
<i>False</i>	False	False

<b>a Or b</b>	<i>a</i>	
<i>b</i>	<i>True</i>	<i>False</i>
<i>True</i>	True	True
<i>False</i>	True	False

<b>a Xor b</b>	<i>a</i>	
<i>b</i>	<i>True</i>	<i>False</i>
<i>True</i>	False	True
<i>False</i>	True	False

<b>a Eqv b</b>	<i>a</i>	
<i>b</i>	<i>True</i>	<i>False</i>
<i>True</i>	True	False
<i>False</i>	False	True

<b>a Imp b</b>	<i>a</i>	
<i>b</i>	<i>True</i>	<i>False</i>
<i>True</i>	True	True
<i>False</i>	False	True

### Let's try! Corner

Suppose the test score of a student is uniformly distributed in the interval [0,100]. We wish to simulate the student's score and output the corresponding result as below.

Score	Result
[0, 40)	Fail
[40, 70)	Pass
[71, 90)	Good
[90, 100]	Excellent

- 1 Open Visual Basic Editor and open Module1. (If you do not have Module1 create a new one in the top left pane.)
- 2 Insert an empty Sub. You can do this by typing the code or insert it from the menu: Insert > Insert Procedure... (You need only type in the name. Keep other options intact.)
- 3 Declare a variable `rnum` with type `Double` to hold the random number generated, and a variable `str` with type `String`.
- 4 Assign a random number to `rnum`. This is done by using the function `Rnd()`, which will return a different number of range [0,1) each time you execute the function. The statement should be:  
`rnum = Rnd`
- 5 We want to output the score to the user. We first construct the string and hold it temporarily in the variable `str` like the following.  
`str = "You got " + CStr(Round(rnum * 100)) + " marks." + vbCrLf`
- 6 Construct the if-then-else statement.  

```
If rnum < 0.4 Then
    MsgBox str + "Fail!"
ElseIf rnum < 0.7 Then
    MsgBox str + "Pass!"
ElseIf rnum < 0.9 Then
    MsgBox str + "Good!"
Else
    MsgBox str + "Excellent!"
End If
```

## 2-10-2: Select Case Statement

This is an alternative to If...Then...Else statement. The syntax is:

```
Select Case testcondition
  [Case expressionlist
    [statements]]
  [Case expressionlist2
    [statements2]]
  ...
  [Case Else
    [elsetatements]]
End Select
```

You place the variable to be tested in *testcondition*, and then you specify test values in expression lists. You can specify multiple values or ranges in expression lists.

Here is an example select statement to illustrate the use of expression lists.

```
Select Case number
  Case 1, 2, 3
    MsgBox "1, 2, or 3"
  Case 4 To 10
    MsgBox "Between 4 and 10"
  Case Is > 10
    MsgBox "Greater than 10"
  Case Else
    MsgBox "Less than 1"
End Select
```

For the line `Case Is > 10` the keyword `Is` is used to replace the variable `number`. In fact you can also use `number` in place of `Is`.

### Let's try! Corner

We try to change the last Let's try! Corner problem using a select case statement. Try yourself.

## 2-11: Loops

It is the power of doing loops to persuade you to write programs. You can perform a certain task several times with loops while certain conditions satisfy. There are several forms of loops you can choose in VBA.

### 2-11-1: For Loops

There are two types of for loops in VBA: For...Next loop and For Each...Next loop.

#### 2-11-1-1: For...Next Loop

The syntax for a For...Next loop is:

```
For counter = startValue To endValue [Step nStep]  
    [statements]  
Next counter
```

The For...Next loop is mainly used to do a block of statements countably many times. You need to specify a variable in place of *counter*. You also specify *startValue* and *endValue* (numbers, not necessarily integers) and optionally *nStep*, so that the following three things happen:

- 1 Before entering the For loop, the counter will be assigned the *startValue*.
- 2 After each execution of the block of statements in the For loop, the program increments counter by *nStep*. If *nStep* is not specified, counter will be incremented by 1.
- 3 After the increment, the program checks if *counter* > *endValue*. If it is the case, the loop ends immediately, and the program will go on executing the program, starting from the statement after the Next *counter* line. If not, the loop continues.

If you want to quit the for loop some point between the statements in the for loop, you can use the statement Exit For to do so. Usually you will use it with If...Then...Else statements. For example,

```
For i = 1 To 10  
    If i > 5 Then Exit For  
    MsgBox i  
Next i
```

#### Let's try! Corner

Suppose we are trying to let the user input *n* numbers by input box so that the computer will calculate the sum and the mean of the *n* numbers.

- 1 Create an empty Sub.
- 2 Declare four variables: *i* (Integer, used as a counter in the for loop), *n* (Integer, used as an input of the number of numbers to be provided), *sum* (Double, used for summing the numbers inputted) and *numInput* (Double, used as a temporary storage of the inputted number)
- 3 Ask an input of *n* by using an input box.
- 4 Initialize the variable *sum* to 0. (Your initial sum before adding the numbers should be 0.)
- 5 Write a For loop:  
For *i* = 1 To *n*  
Next *i*
- 6 Within the for loop, ask the input of a number from the user, and put it in *numInput*. Add *numInput* to the *sum*.
- 7 Below the for loop, show two message boxes, one outputting the sum and another outputting the mean.

**Let's try! Corner**

Write two procedures to calculate the following sums, using input boxes to ask for values of  $n$ :

1.  $\sum_{k=1}^n k^2$

2.  $\sum_{k=1}^n I(k \text{ is even})e^{-k}$ , where  $I(k \text{ is even}) = \begin{cases} 1 & \text{when } k \text{ is even} \\ 0 & \text{when } k \text{ is odd} \end{cases}$

### 2-11-1-2: For Each...Next Loop

The syntax for a For Each...Next loop is:

```
For Each element In group
  [statements]
Next element
```

This is useful to loop through each element in **group**. The variable **group** should be a collection or an array. You will study collections and arrays later. Basically the working principles are the same as the For...Next loop:

- 1 Before the execution of the for loop, **element** will be assigned the first element in **group**.
- 2 After all statements have been executed, the program checks if there are more elements in **group**. If there are, the next element will be assigned to **element** and the statements inside the loop repeat. If no element is available, the loop finishes and the next statement after **Next element** line will execute and continue.

### 2-11-2: Do Loops

Do loop allow you to specify conditions where you want the loop end. There are 5 types of Do loops. The syntax for the simplest one is:

```
Do
  [statements]
Loop
```

As usual you can put many statements in *statements*. This type of Do loop will continue to loop through the statements inside infinitely. Like For loops, you will have a chance to stop the Do loop by using the statement `Exit Do`. (You can use it for all types of Do loops.)

Another four are similar:

```
Do While condition
  [statements]
Loop
```

```
Do Until condition
  [statements]
Loop
```

```
Do
  [statements]
Loop While condition
```

```
Do
  [statements]
Loop Until condition
```

They are pretty self-explanatory. The difference between using `While` and `Until` is that the loop will continue if the condition is `True` with `While`, and `False` with `Until`. The difference between putting the condition part before the statements and after the statements is that if you put it beforehand, the condition will be checked beforehand, and vice versa. So the statements in the loop will be guaranteed to run once if you put the loop condition at the back.

#### Let's try! Corner

We have written a Sub to ask users to calculate the sum and the mean of  $n$  numbers. Try rewriting it using a Do loop. This time do not ask the user to input  $n$ . Instead, the program should detect the end of the series of numbers when the user clicks the Cancel button in the input box.

### **2-11-3: While Loop**

It is the synonym of Do While...Loop.

```
While condition
  [statements]
Wend
```

```
Do While condition
  [statements]
Loop
```

The above two loops are equivalent, except that you cannot exit While loops using `Exit` statement.