VBA for Microsoft Excel – session 2

Table of Contents

1 Two Inbuilt functions: MsgBox and InputBox

`Msgbox` and `InputBox` are functions that will work in any Visual Basic environment. For example, they work the same in Outlook or Word.

MsgBox is a function that communicates information to the user while the macro runs. The function takes at least one argument, namely the message, referred to as the Prompt, which must be enclosed in two double quotes. You can embellish the box with other information such as a Title or different types of buttons.

1.1 MsgBox - the basic function

`MsgBox` displays text in VBA. In its most simple form, it accepts one argument, namely the text to be displayed. The syntax is:
`MsgBox` *`String`* where *`String`* is a string of alphanumeric characters, i.e., text:

```
Sub HelloWorld()
    MsgBox "Hello World"
End Sub
```

Note that text is marked by double quotation marks.

For numbers, the quotes are not necessary. So:

```
Sub HelloWorldNumbers()
   Msgbox 42
End sub
```

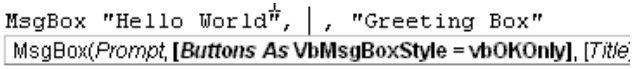
is perfectly legitimate

1.1.1 MsgBox - with more arguments

The Message Box above is a little plain. You can embellish the box with other information such as text in the Title bar of the Message Box or different types of buttons. If so, you can supply more *Arguments* to the `MsgBox` function. Arguments are separated by a comma

```
Sub HelloWorld2()
   MsgBox "Hello World", vbOKCancel, "Greeting"
End Sub
```

In the VB Editor, as you type the MsgBox statement a yellow box appears to guide you – this is called the Quicklist.



In the **MsgBox** example the second argument determines what buttons appear in the message box. After you type the first argument and have typed the comma, a list of possible values for the second argument appears. By default you would get the **OK** button, which is all we want for now, so you can leave the argument blank and the message box will have an OK button.

The third argument is **Title**, which is the text to appear in the Title bar. It makes the box look more finished, but isn't absolutely necessary. A list of possible values does not appear because you decide what the title should be.

```
MsgBox "Hello World",, "Greeting Box"
```

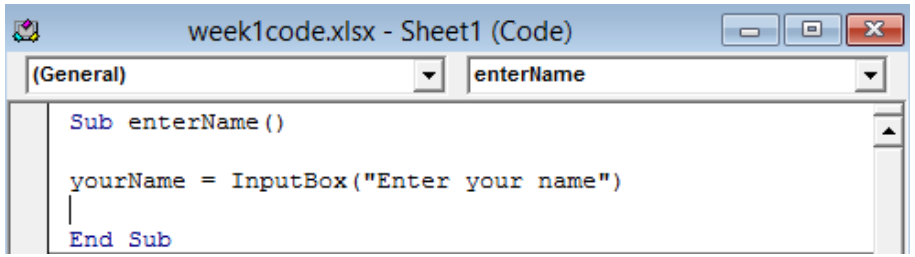
The `Buttons` argument is left blank (between the commas) so the default value, `vbOKOnly` is assumed. This displays an OK button only

1.2 InputBox

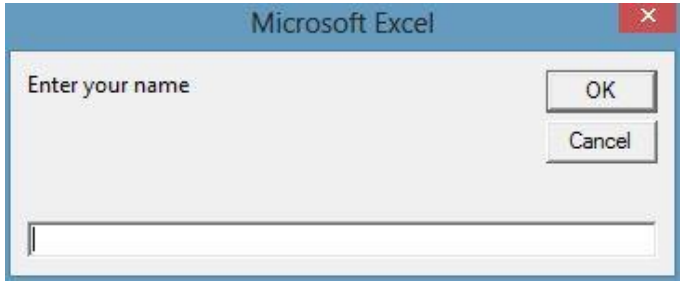
An Input Box prompts the user to enter some information, which can then be manipulated by a Visual Basic program. It returns what the user types. The syntax is as follows:

```
variable = InputBox (Prompt)
```

The following asks the user to type in his/her name.



- The Variable, `yourname`, holds the return value (more on this later)
- The argument must be in brackets because the InputBox here is returning a value using the assignment statement (equal sign)
- `Prompt` is what the user sees:



.

There are a number of other, optional arguments (shown in square brackets)

```
variable = InputBox (Prompt), [Title, Default, XPos,
YPos, HelpFile, Context])
```

You can also specify a title, default entry text as well as X and Y co-ordinates on the screen.

```
(General)                                    enterStudentName
Sub enterStudentName()
studentName = InputBox("Enter your name",_
       "City University", "John Smith", 5000, 3000)


End Sub
```

The second argument is the text that appears in the Title bar of the InputBox window.

- The third argument is a default value for the user's input, here **John Smith**. The user sees this and can simply click OK to return the default value.
- Note the use of space and underscore to continue the statement on the next line

The result on screen looks like this:

1.3 Making the text in InputBox/Msgbox appear over separate lines

```
Msgbox "Pay attention: " & vbNewLine & "Don't forget"
```

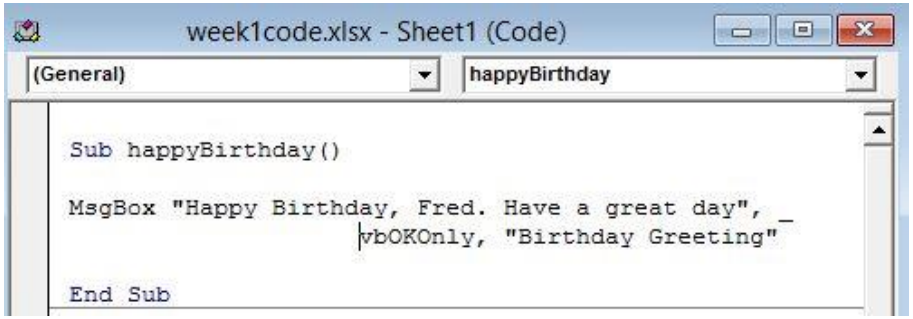
The `MsgBox` display text has three elements: the first line in quotes, the second element is the `vbNewLine` constant and the third argument is the second line of text.

The & (Ampersand) character is the recommended Concatenation operator: it joins strings and variables.

You can also use `vbCr`, `vbLf` and `vbCrLf`; `vbNewLine` should work on all operating systems.

1.4 Splitting a line of code over several lines

In the Visual Basic *Editor*, you type one statement per line, which is a problem with long lines of code. You can easily split a statement by ending a line with a space and underscore character. For example,



This doesn't change how the message appears to the user.

1.5 Another way of using Arguments

You can use *explicitly named arguments* in your code. For example, the line

```
MsgBox "Hello World",, "Greeting Box"
```

could be written:

```
MsgBox Prompt:="Hello World", Title:="Greeting"
```

The code works exactly the same way, it's just that the argument names are included. Though there is more typing involved, the advantage is that the code is easier to read. Another characteristic is that now the order of the arguments doesn't matter, so you could just as well type:

```
MsgBox Title:="Greeting", Prompt:="Hello World"
```

The syntax is for arguments and their values is:
ArgumentName := *Value*

This is different from the syntax for properties where there is no colon.

1.6 InputBox and MsgBox combined by using a variable

We will write a macro that asks the user for their name, and says "helloto the person whose name was entered using InputBox.

The program is as follows:

```
Sub hello()
Dim username As String ‘declare variable
username = InputBox("Please enter your name")
MsgBox "Hello " & username
End Sub
```

- `Dim` is the keyword used to declare a variable
- In line 3 the variable is assigned the value returned by the `InputBox` function.
- Line 4 messages the greeting concatenated with the value of the variable

Just an Excel function takes an argument and returns a value, so does a VBA function. Think of the Excel `Average` function:

```
=Average(B3:B22)
```

The argument for `Average` is a list of numbers, or, more usefully, a range of cells containing numbers – here `B3:B22`. The return value appears in the cell.

With `InputBox`, the string (here, "`Please enter your name`") is the argument. The return value is assigned to the `username` variable.

2 Variables

Variables are storage containers for data used within a macro (or any computer programme). Typically the programme manipulates the data in some way, for example performing mathematical calculations or displaying the data on screen. As the name implies, it is possible to change the value as the programme.

2.1 Variable names

Variables must have a name.

- Valid variable names can contain text or numbers but they must use a letter as the first character.
- You can't use a space, period (.), exclamation mark (!), or the characters @, /, &, $, # in the name.
- Names cannot exceed 255 characters in length (short and to the point is best)

Examples of valid variables are:

```
City
CustomerCity
Customer_City
Sham69
```

Examples of invalid names are:

```
4thjuly         starts with a number
/               invalid character
hello wally     contains a space
```

2.2 Variable types

There are different types of variable. You should choose the appropriate type for a given data. For example, text is usually declared as a string. Numbers are declared as either `Integer` (whole number), `Long` (very large whole number), or as `Single` (a real number e.g. 4.1702), `Double` (a large real number). There are other types too, including `Currency` and Date.

There are different types of variable primarily because some types use less memory than others, so choosing the right type makes your program run more efficiently.

2.2.1 Numeric variables

VBA type	Range of Values
Byte	0 to 255
Integer	-32,768 to 32,767
Long	-2,147,483,648 to 2,147,483,647
Single	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double	-1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807

2.2.2 Other variable types:

VBA type	Range of Values
Boolean	True or False
Date	1/1/100 to 31/12/9999 VBA defaults to U.S. format
String	Up to approx. 2 billion characters (65,400 if fixed length)
Object	Any Object reference
Variant	Number: same as Double String: same as String

If you don't choose a type, VBA uses the `Variant` type. This may seem easier, but there are disadvantages:

- variants use more memory so code runs more slowly
- if a numeric value is interpreted as a string, calculations could give weird results e.g., `1 + 1` would give `11`

2.3 Declaring and assigning variables

In standard programming practice, the following happens to variables:

- a variables is declared ("this variable now exists")
- a variable is usually assigned a value at the beginning. A variable normally takes different values during the running of the programme

2.3.1 Declaring a variable

To declare a variable, you use the keyword `Dim` (`Dim` stands for "`Dimension`" referring to the size of the variable). The optional keyword `As` assigns a data type. If you omit the keyword `As` the variable will be a variant

Examples:

`Dim CustomerName as String`	*for text/numbers in double quotes*
`Dim Hours as Byte`	*for small whole numbers*

`Dim Attendance as Integer`	*for whole numbers*
`Dim DistanceFromSun as Long`	*for larger whole numbers*
`Dim Vat_Rate as Single`	*for real numbers with a franctional part*
`Dim EuroBond as Double`	*for larger and more precise real numbers*
`Dim NewFinYear as Date`	*for dates*
`Dim Revenue as Currency`	*for currency values – you can use 15 digits before the decimal point, 4 digits after the decimal point*
`Dim CreditGood as Boolean`	*for True or False*

It is possible to declare several variables in one line of code but note that

```
Dim x, y, z as integer
```

declares `z` as an integer; `x` and `y` as variants

The code should be instead:

```
Dim x as integer, y as integer, z as integer
```

2.3.2 Assigning variables

Simple! Type the variable name, then an equals sign then on the right hand side of the equals type a suitable value.

Examples:

```
x = 2
```
the value 2 is assigned to x

```
2 = x
```
is nonsense because 2 is a literal number, not a variable

```
SalesRegion = "North West"
```
you must use double quotes when assigning string variables

String variables are case sensitive; "`North West`" and "`NORTH WEST`" are not equivalent values.

```
NextFinYear = #1-April-2013#
```
Use # to enclose the value of a date

NB the format of a date variable defaults to American (can be confusing for most of us over here!)

It is possible to assign the value of one variable to another

```
y = x
```
a variable `y` *is assigned the same value as a variable* `x`

```
DeliveryDate = OrderDate
```
assigns the value of the variable `DeliveryDate` *to be the same as* `OrderDate`

A variable can store the result of a calculation, e.g.,

```
OrderTotal = Subtotal * VatRate
```

You can do calculations involving dates, e.g.

```
OrderDate = #9-March-2012#
DeliveryDate = OrderDate + 3
```

the above two lines would put into the variable `DeliveryDate` the date that is three days after the value that is in `OrderDate`

Here's how to increment the value of a variable:

```
OrderNo = OrderNo + 1
```
increases the value of a variable called OrderNo by 1

this kind of operation crops up quite frequently in programming.

NB If you declare a numeric variable then its value is 0 until it is assigned. If you declare a string variable its value is the empty string "" until it is assigned.

2.3.3 Arithmetical Operators

Operator	Operation	Example	Answer
+	Add	5+4	9
-	Subtract	7-2	5
*	Multiply	3*5	15
/	Divide	10/4	2.5
\	N1 \ N2 - integer division	15\4	3
Mod	X Mod Y returns remainder	10 mod 3	1
^	X ^ Y gives the value of X raised to the power of Y	2^3	8

2.3.4 Concatenation Operator

The concatenation operator **&** (shift + 7) is used to join expressions together e.g. a number to text. For example:

```
MsgBox "You will be paid " & payrate & " per hour"
```

"**&**"is able to concatenate strings **with** numeric variables, so you must use it if you want to combine a string and a number.

The plus operator (+)

There is also the "**+**" operator but this can only concatenate one string to another; if you use it to concatenate a number to text you will get a run-time error (so in my opinion it's best never to use it!)

2.4 Examples

```
Sub Example1()
'declare variables
  Dim x as integer, y as integer, z as integer

' assign values to variables
  x = 3
  y = 4
  z = x + y

' output result
  Msgbox z
End sub
```

This program is very limited because the numbers are *hard-coded* (fixed within the program). You could read in a number from an input box.

```
Sub EnterNumbers ()
  Dim num1 as Integer
  num1 = InputBox("Enter number under 5000", _
                  "Enter numeric data")
  num1 = num1 * 2
  MsgBox "The number multiplied by 2 is " & num1
End Sub
```

More typically variables' values are read in from a worksheet.

2.5 Examples using cells

You can refer to a cell using the range object. We'll talk about this more in later classes, but for now, here is how to refer to the contents of a cell.

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

Range("A1") is the object and *value* is a *property* of that object.

So you could have:

```
Dim EuroRate as Single
EuroRate = Range("B4").Value
```

It's just as common to work directly with cells on the worksheet so you could skip the use of variables. Each of the following lines is a valid statement:

```
Range("A1").Value = 4
Range("A1").Value = Range("A1").Value * 2
Range("A1").Value = Range("B1").Value
```

where

Range("A1") is the cell A1 on the currently open worksheet.

Here is how to put a value directly into cell A1 via an input box:

```
Range("A1").Value = InputBox("Enter a number")
```

2.6 ActiveCell

The cell that is selected on the current worksheet can be referenced as `ActiveCell` This reference is widely used. For example,
`ActiveCell.Value = 99`
writes 99 into the active cell in the active worksheet, irrespective of its address. Programs that are written to be more generally useful invariably use references to `ActiveCell`.

This is rather like using `Selection.Value = 99`, except that a selection can be multiple cells, but there is only one active cell.

2.7 Do variables have to be declared?

In VBA is it possible to use variables without declaring them? It's possible, but not desirable. It all depends on whether Option Explicit is present at the top of the module. Let's say the table below is an overview of a module sheet.

```
Option Explicit
-----------------------------------
Sub Macro1()
  …… (code here)
End Sub
-----------------------------------
Sub Macro2()
  …… (code here)

End Sub
```

If `Option Explicit` is missing then you do not have to declare variables. That means that the following program – with no declaration - will work:

```
Sub test()
  x = 5 + 4
  MsgBox "5 plus 4 is " & x
End Sub
```

This may seem convenient, but in the long run will cause problems as mis-spellings will not be automatically flagged up. With Option Explicit at the top of the module, you cannot use a variable unless it has been declared. A mis-spelt variable will by definition be undeclared and the error will be brought to your attention.

To work with Option Explicit by default:

1. From the VBE editor, choose Tools, then Options
2. In the Editor tab, check the checkbox Require Variable Declaration

3 Some text functions

3.1 Ucase/LCase

You have already seen UCase and LCase in relation to cells. They can also work in other contexts.

```
yourName = InputBox("Type your name")
MsgBox UCase(yourName)
```

asks the user to type their name and returns it in capitals.

3.2 Len

Len finds the number of characters in a string.

```
Option Explicit
Sub textfunctions()
Dim s As String
s = InputBox("please type your name")
MsgBox "Your name has " & Len(s) & " characters"

End Sub
```