A (Very) Brief Introduction to VBA

Visual Basic (VB) and Visual Basic for Applications (VBA) look nearly the same. We won't really be able to see the difference in this course, but since we are always writing code in Excel (in the Visual Basic Environment, or VBE), we are, technically, using VBA, the interpreted language that is Microsoft-application specific.

Comments

Comment lines begin with "Rem" (outdated, short for "remark") or a single apostrophe. In fact, an apostrophe anywhere on the line begins a comment (unless it is in quotation marks), and the comment continues to the end of the line.

```
x = x + 1    'Increment x
```

Data Types

VBA uses a few basic "built-in" data types for variables. All variables are either built-in data types, user-defined data types, objects, or are arrays of these types. Some of the most commonly used built-in data types are:

Name	Size	Values
`Boolean`	2 bytes	{True, False}
`Integer`	2 bytes	-32768 to 32767
`Long`	4 bytes	-2147483648 to 2147483647
`Single`	4 bytes	about +/- 3.4E38 down to about +/- 1.4E-45
`Double`	8 bytes	about +/- 1.8E308 down to about +/- 4.9E-324 (more significant digits also)
`Date`	8 bytes	01/01/0100 to 12/31/9999
`String`	10 bytes +	about 2 billion characters (one byte per character)
`Variant`	16 bytes	??? anything you want it to be

Declaring Variables

ALWAYS DECLARE YOUR VARIABLES. For example, if you wish to use a variable, x, as a long integer, you declare it at (or near) the beginning of your subroutine with a "Dim" (short for "dimension") statement:

```
Dim x As Long
```

This tells the VBA interpreter that x will always be an integer-valued variable, and that you need 4 bytes to store its possible values.

You can "require" yourself to declare all variables by starting each code module with

```
Option Explicit
```

This will then generate an error if you ever refer to a variable you haven't declared. (The default, which is *terrible*, is that VBA will assume it is a new variable of type Variant!)

Arrays

If you want an array y indexed from 1 to 10, use a Dim statement such as:

```
Dim y(1 To 10) As Double
```

The Dim statement for arrays sets the lower and upper bounds on the index. If no lower bound is set, it is assumed to be zero. So:

```
Dim y(10) As Double
```

declares y to be an array of 11 (!) doubles, accessed as `y(0)` through `y(10)`.

Multidimensional Arrays

You can have arrays with more than one index:

```
Dim range(1 To 10, 1 To 20) As Double
```

You can then access entries of y as, say, `y(3,2) = 1.04562`

Dynamic Array Sizes

If you don't know how big y will need to be, start with

```
Dim y() As Double
```

(which says y will be an array of doubles, but doesn't promise anything about size) and then, later, if you find out you need ysize entries in the array, you can "Redimension" it:

```
ReDim y(ysize) As Double
```

If you want to use a variable (as opposed to a constant) as an array size, you *must* use ReDim. This is equivalent to dynamic memory allocation in other languages (e.g., malloc() in C). ***(This one concept cost me about a week of my life. The online help is not clear about this at all!)*** You can ReDim as many times as you want, changing the size (and even the number of dimensions) each time. If the array gets bigger, you can use something like:

```
ReDim Preserve y(ysize + 100)
```

to save all of the original values. Only do this if you know you need EVERY value in the old array, since this can be fairly expensive, run-time wise; VBA has to copy the entire array to a new location.

Flow Control

VBA has all the basic flow control structures that other languages have:

If-Then

```
If condition1 Then
    ...
[ElseIf condition2 Then]
    ...
[Else]
    ...
End If
```

For-Next

```
For counter = start To end
    ...
    [Exit For]
    ...
Next [counter]
```

Do Loops

```
Do [While condition | Until condition]
    ...
    [Exit Do]
    ...
Loop [While condition | Until condition]
```

And a few others, including **Select-Case** and **GoTo**.

Sub and Function Procedures

```
Sub subName(arg1 As type1, arg2 As type2, …)
     Dim ...
     ...
End Sub

Function funcName(arg1 As type1, arg2 As type2, …) as type3
     Dim ...
     ...
     funcName = value
     ...
End Function

x = funcName(5, 3.2, "Charlie")
Call subName("Bravo", 2)
subName arg1:="Bravo" arg2=2
```

↑ can be variants if you don't know what they will pass you

Variable Scope

Variables that are dimensioned within a procedure can only be used by the procedure. To allow them to be used by all procedures in a module, declare them in the code module *before any procedures*. They act like global variables, and their values will persist as long as the worksheet is open. If you want variables to be visible to other modules, declare them as `Public` module variables. Finally, if you want a local variable to retain its value, declare it as `Static`.

Constants

Constants can be declared as:

```
Const avogadro As Double = 6.02E23
Public Const pi As Single = 3.12
```

Of course, you might want to use a better value for pi, or use a different name for 3.12...

User-defined Data Types

The user-defined data types are the same as records in other languages. You mash a few useful data types together, and give yourself a way to access the components, and then you can have an array of these records that store more than one piece of information per entry.

```
Type ArcType
     Tail As Long
     Head As Long
     Cost As Long
     Capacity As Long
End Type
```

Now, if we want to declare an array "arcs":

```
Dim arcs(1 to 10000) As ArcType
```

And we can refer to the arcs, and their components, as:

```
arcs(10).Tail = 7
arcs(342).Cost = arcs(341).Cost + 2
arcs(7) = arcs(10)   'Really useful: this copies all fields!
```