Visual Basic - Introduction

Introduction

Visual Basic for Applications (VBA) is the programming language used in Excel. With it you can automate tasks, add custom features and functions, and even create complete applications (-this is too advanced for this introductory course).

In Excel tasks are automated by macros – a sequence of instructions written in VB that tell Excel what to do. To facilitate their creation Excel provides a feature called *the macro recorder* - which writes macros for you! It stores the keyboard actions and commands you use. Later you can play back, or run, the macro to repeat automatically the actions. This is probably the easiest way to get into VB and macro programming. You'll soon discover how useful macros can be and want to modify the recorded code so as to improve it or to use it in a slightly different situation.

The more you know about Excel, the better prepared you will be for macros and VB. Most macros perform a sequence of actions in Excel. Most macro instructions are equivalent to commands or actions in Excel. Those that are not are, however, among the most useful! Nevertheless the statements and functions used to write instructions are easier to understand if you know the features they represent in Excel.

VB can seem overwhelming especially if you have no experience with a programming language. To overcome this we can use the Excel macro recorder facility to see how Excel actions translate to VB instructions and vice versa. One tactic is to write part of your macro by directly typing VB statements into the macro, then record part of it using the macro recorder (often because you don't know how to write that part in VB!) and merge the parts. Recording part of the macro is often faster and easier than writing out the VB statements. In a VB module you can type a keyword and, with the insertion point somewhere in the word, press F1 to display help on that keyword. For many keywords there is given an example of its use which you can copy and paste into your own macro.

Programming style

It is essential that there should be lots of comments in a macro explaining what the various sections of the code are doing. These comments greatly help debugging the code. You don't want to have to work out each time what it is doing. If after a long time you return to code without comments you are unlikely to remember what the macro does in detail, if at all. An apostrophe (') introduces comments into the code. Statements to the right of the ' on the same line are not executed.

It is useful to adopt a *programming style* for ease of reading of the code. We shall capitalise names of macros and user-defined functions. Macro names and functions cannot include spaces so if a name consists of more than one word the other words have their initial letters capitalised, e.g. TaxReturns, CourseWork.

Arguments and variable names appear with initial letters in lowercase – to distinguish them from macros, functions, properties, methods and object names.
Keywords appear with the initial lette r capitalised.
Built-in constants appear with lowercase "xl" or "vb".
Control blocks and statements in **Sub** and **Function** procedures are indented within the code.
The line continuation character, the underscore, (_), indicates that the code continued from one line to the next is part of the same logical statement.

```
Function Discount(quantity, price)
‘ calculates discount
‘ discount is only given for orders of more than
‘ 100 widgets
    If quantity >= 100 Then
        Discount = quantity * _
                   price * 0.1
    Else
        Discount = 0
    End If
    Discount = Application.Round(Discount,2)
End Function
```

Modules and procedures

These are the basic structures of a VB programme. We need to break complex tasks into simple procedures, to invoke one procedure from within another procedure and to pass data between them.
A programme made from small (reliable) components has many advantages over one built on a single structure. A well structured modular programme is

- easy to write because complex problems are broken down into a series of discrete, easy-to-understand units,
- easy to debug as each task is completed in one procedure and so it's easy to isolate the source of the error,
- efficient and easy to modify as code that accomplishes a common task appears in only one procedure that's called many times, rather than being duplicated many times throughout a single structure.