Practical 2 – An Introduction To Coding

Aims

This tutorial is chiefly designed to introduce some basic operators available in Visual Basic and to practice variable declaration.

1 Naming variables

- With most languages variable names must begin with a letter and be followed by any number of alphanumeric characters. For instance, `A5` is a valid variable name, but `5A` is not.
- The maximum size allowed for variable name is 255 characters.
- You are not allowed to use key words as variable names. A keyword is a reserved word or symbol recognised as part of a programming language.
- In some languages variable names are not case sensitive, in others they are.

  The VB editor automatically changes variable names to the same case as their first occurrence. For example, if a variable is named `todaysDate` and then later another variable `todaysDATE` is typed, the VB editor automatically changes this second variable to `todaysDate`.

Variable names should also be **self-commenting**, or describe their contents. Anyone reading the code will have no idea what a variable named `A5` contains. Compare that with a variable named `todaysDate`, which is self-explanatory.

There are a number of different ways to write a variable name, for example `lastName`, `LastName`, or `Last_Name`; using lower and uppercase characters or an underscore to distinguish the words.

Some organisations also use naming conventions. For instance all forms might begin with a lower case `frm` e.g. `frmName`. Similarly all integers begin with lower case `int` e.g. `intCount`. (You can look at http://support.microsoft.com/default.aspx?scid=kb;EN-US;q173738 for one naming system called *Hungarian Notation*.)

2 Declaring Variables

You are expected to explicitly declare variables, although VB does *not* require you to. However it is important to do so for the following reasons:

1. It allows other programmers reading your code to easily determine the type of data stored in the variables and the ranges of the variables.
2. It provides greater control over the memory required by your program. Unless otherwise specified, VB assumes a variable is of the `Variant` data type and 16 or more bytes will be reserved for its storage. You can save considerable memory by explicitly assigning a specific data type to each of your variables.
3. It avoids confusion between variables of the same name in different sections of your code.
4. It encourages use of self-commenting variable names.
5. It gets you in the habit of declaring variables before their use, making it easier to change to other computer languages.
6. It provides an opportunity for additional commentary.

VB can be set up to ensure that you explicitly declare all of your variables by selecting Tools/Option and looking under the Editor tab. Make sure that there is a check mark in the Require Variable Declaration check box.
Alternatively, simply type **`Option Explicit`** on a single line at the top of your code before all procedures.
Most languages allow multiple declarations such as this.

```
Dim var4, var5, var6 As Double
```

But DON'T FALL INTO THIS TRAP! Only `var6` is declared as a double precision real number. `var4` and `var5` are of the `Variant` data type (the default data type). You are also not able to comment each variable separately.

3 First Program

We repeat the first program from the lecture. Follow these description below.

1. Load Visual Basic, select **"Standard EXE" when the screen prompts you**

2. Locate the **CommandButton** icon on the **toolbox to the left** of the screen

3. Click, and then drag a square on the grey "`Form1`"

4. Locate the **properties window**

5. Make sure the **"Alphabetic" tab** is selected

6. Locate the **"(Name)" property** – which should be at the top of the list

8. Delete what's there, and type (use the upper/lower case letters as shown) :

```
cmdGo
```

9. Double click on the "Caption" property (again, in the Properties window)
   - this property will change what appears on the button when the program is running
   - it is **NOT** the same as the "Name" property

10. Type some meaningful text to appear on the button (e.g. "Go")

11. **Double click the actual command button** on the form, and a window should appears containing :

```
Private Sub cmdGo_Click()
End Sub
```

12. **Between the lines** beginning "`Private Sub ...`" and "`End Sub`" type the following line of code (be *very precise* – everything is critical except the text between `"` symbols, which can be practically anything) :

```
Dim x As Integer
Dim message As String
Dim title As String
title = "Pay attention"
message = "Hello World !"
x = MsgBox(message, vbCritical, title)
```

13. Now, **locate the "Run" button** ▶ on the toolbar underneath the menu bar (you can also find it under "Run / Start"). The program is now compiled and executed immediately.

14. Once you've finished experimenting, locate and click the "Stop" button ■ . It's just alongside the "Play" button and will halt the program.

Note `vbCritical` is a Visual Basic constant. A table of other constants for the message box function is given at the end of this document.

4 Saving the Project

The program you've just written is part of a so-called "Project" which is the collection of all segments of code, all the forms and all the objects on all the forms. Often, one project has many forms and code segments (as well as a number of other files).

To save a project, you must decide where you put it, and this is important because it's difficult to move a project to another location once it has been created in one location; you must remember to move every single file associated with the "umbrella" project.

You can save projects to one of two locations:

- On your floppy disc (recommended, then you can continue coding during directed study)
- on your University file store (check that you are able to fetch programs from the University store from home)

Follow these instructions to save the project.

1. Go to **File / Save Project**
2. Make sure you **change directory** to your floppy (A:) or a network drive (for central file store)
3. Give the **form a name** such as `first_program` (each form is saved as a separate file with extension `.frm`)
4. **Click "Save"**. Now, you have saved the only component of your project (a single form). You will be **prompted for a name for the whole project**. This means when you open the project again, all the associated files (e.g. forms) will be loaded. Type a filename (perhaps, `first_program`)

This project can now be re-opened in the future. Make sure you have two files on your disc, one called `first_program.frm` and the other called `first_program.vbp`. Note that saving forms and saving projects are two operations. Visit the File menu to see the two actions.

5 Initialising Variables

In the program you typed in each of the variables was assigned a value immediately after it was declared. After any variable is declared, it must then be **initialised,** that is, assigned an initial value.

The default initial value in VB is:

- Zero for numeric variables
- False for Boolean variables
- **null string** (the string containing no characters) for string variables

Although VB automatically initialises variables to these default values in the variable declarations, most other high-level languages do not. In most other languages, after a variable is declared, it contains whatever "junk" value is residing in its assigned memory location. If there is no initialisation, in other high-level languages some unspecified previous value will be used. Even in VB, problems can arise, therefore, *initialise all variables in your program after declaring them*.

Mathematical programming

Just as there are arithmetic **symbols** that denote specific operations to be performed in mathematics, there are equivalent **operators** to denote these operations on a computer.

Mathematics	Math Example	Visual Basic	VB Example
Exponentiation	5^2	^ (caret)	`5^2`
Negation, –	–5	– (minus sign)	`-5`
Multiplication, ×	5 × 2	* (asterisk)	`5*2`
Division, ÷	5 ÷ 2	/ (solidus)	`5/2`
Addition, +	5 + 2	+ (plus sign)	`5+2`
Subtraction, –	5 – 2	– (minus sign)	`5-2`
Integer division		\ (reverse solidus)	`5\2`
Modulo division		`Mod`	`5 Mod 2`

Table 1 Visual Basic arithmetic operators

Integer division and **modulo division** may be mathematical operations unfamiliar to you. Integer division, denoted by a reverse solidus (`\`) in VB, is used to divide two numbers and return only the integer part of the *quotient*; any fractional portion is simply truncated or ignored. For example, the result of `5\2` is `2`. While `5` divided by `2` equals `2.5`, it is only the integer part of the quotient, `2`, that is the result of integer division. The fractional part of the quotient is ignored.

Modulo division, denoted by the `Mod` operator in VB, is used to divide two numbers and return only the integer portion of the *remainder*. For example, `5 Mod 1.7` equals `1` since `5` divided by `1.7` equals `2` with a remainder of `1.6`. The integer part of the remainder, `1`, is the result of modulo division.

5.1 ORDER OF OPERATIONS

Mathematical operations are not evaluated from left to right. Instead, mathematical operators follow certain **precedence rules**, or an **order of operations**, that are listed below.

1. Exponentiation (`^`)
2. Negation (`-`)
3. Multiplication and division (`*`)
4. Integer division (`\`)
5. Modulo division (`Mod`)
6. Addition and subtraction (`+`, `-`)
7. String concatenation (`&`)

When multiplication and division occur together in an expression, then each operation is evaluated as it occurs from **left to right**.

Likewise, when addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from **left to right**.

The string concatenation operator (`&`) is not an arithmetic operator; it is performed at the end, after all the arithmetic operators.

Parenthetical expressions may be used to override predefined precedence rules.

Second Program

We will try out some of the maths functions.

1. Load a new project, **"Standard EXE"**.
2. Click a **Command Button** from the toolbox and then drag it onto the form.
3. Likewise with a **label box** and two **Text Boxes**, and drag them onto the form.
4. Change the **"Name" property** of the
   4.1. form to `frmMaths`
   4.2. command button to `cmdGo`
   4.3. label to `lblOutput`
   4.4. text boxes to `txtInput1` and `txtInput2` (delete the labels that come with them).
5. Change the "Caption" property of the
   5.1. form to `Maths Form`
   5.2. command button to `Go`
   5.3. label to `Output`
6. Change the "Text" property of the
   6.1. text boxes to `Input1` and `Input2`
7. **Double click the actual command button** on the form, and a window appears containing:
   ```
   Private Sub cmdGo_Click()
   End Sub
   ```
8. **Between the lines** beginning "`Private Sub ...`" and "`End Sub`" type the following lines of code:
   ```
   lblOutput.BackColor = vbRed
   lblOutput = txtInput1 + txtInput2
   ```
9. Now run the program, put numbers into the text boxes and press the Go button.
   9.1. The correct output should have been the sum of the two numbers on a red background.

   What was actually displayed was the numbers concatenated together, since the text box reads in the numbers as a **character string**. To read the number as *numerals* from a text box we must change the string that is typed to **literal** numbers and we do this using a function called `Val(`*string*`)` which converts the numbers contained in the *string* to a numeric value. Thus you will need to change your program to

   ```
   lblOutput = Val(txtInput1) + Val(txtInput2)
   ```

   9.2. Now try this out again. It works!

   Strictly speaking the answer should be turned back to a string before we assign it to a label. However, VB assumes this is what you are trying to do

and will do it for you. To make explicit the intention of the program, programmers do not usually use the + to concatenate strings (as it can be confused with the add operator); they use the alternative sign for concatenation `&` (ampersand).

9.3. Now, change the mathematical operator, changing the background property of the output box, using Table 1 above.

9.4. Experiment with Tables 2 and 3.

Constant	*Value*	*Description*
vbBlack	&H0	Black
vbRed	&HFF	Red
vbGreen	&HFF00	Green
vbYellow	&HFFFF	Yellow
vbBlue	&HFF0000	Blue
vbMagenta	&HFF00FF	Magenta
vbCyan	&HFFFF00	Cyan
vbWhite	&HFFFFFF	White

Table 2 Visual Basic colour constants

Math Functions	
Abs(x)	**Returns the absolute value of x**
Atn(x)	**Returns the angle corresponding to the arctangent of x in radians**
Cos(x)	**Returns the cosine of angle x where x is in radians**
Exp(x)	**Returns the value e^x where e is approximately 2.718282**
Fix(x)	**Returns the integer part of x. If x is negative, it returns the next integer greater than or equal to x**
Int(x)	**Returns the integer part of x. If x is negative, it returns the next integer less than or equal to x**
Log(x)	**Returns the natural logarithm of x**
Rnd(x)	**Returns a random number. See Visual Basic online help for a description of the result based upon values of x**
Sgn(x)	**Returns a 1 if x is positive. –1 if x is negative, and 0 if x is zero**
Sin(x)	**Returns the sine of angle x where x is in radians**
Sqr(x)	**Returns the square root of x**
Tan(x)	**Returns the tangent of angle x where x is in radians.**
Format(x, fmt)	**Returns the number x in numeric format fmt**
	Predefined numeric format names include:
	Format name (**fmt**) — *Description*
	General Number — Display number as is, with no thousand separators.
	Currency — Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator. Note that output is based on system locale settings.
	Fixed — Display at least one digit to the left and two digits to the right of the decimal separator.
	Standard — Display number with thousands separator, at least one digit to the left and two digits to the right of the decimal separator.
	Percent — Display number multiplied by 100 with a percent sign (%) appended to the right; always display two digits to the right of the decimal separator.
	Scientific — Use standard scientific notation.
	Yes/No — Display No if number is 0; otherwise, display Yes.

	True/False	Display False if number is 0; otherwise, display True.
	On/Off	Display Off if number is 0; otherwise, display On.

Table 3 Visual Basic math functions and numeric formats

Note this program has used the default properties of the text box and label objects. The default properties are the most commonly used properties of a visual object. Rewrite the program so that the properties are explicitly referenced. Thus:

```
lblOutput.Caption = Val(txtInput1.Text) + Val(txtInput2.Text)
```

Another assumption VB makes is that the code is written under the event of a control on the same form. If not, you would also have to reference the appropriate form, e.g.

```
FrmMaths.lblOutput.Caption = Val(frmMaths.txtInput1.Text)
                           + Val(frmMaths.txtInput2.Text)
```

which is considerably more typing than

```
lblOutput = Val(txtInput1) + Val(txtInput2)
```

especially if there are a lot visual objects in the application.

6 The MsgBox Function (abstracted from the help file)

Displays a message in a dialog box, waits for the user to click a button, and returns an **Integer** indicating which button the user clicked.

Syntax

MsgBox(*prompt*[, *buttons*] [, *title*] [, *helpfile*, *context*])

The **MsgBox** function syntax has these named arguments:

Part	*Description*
prompt	Required. String expression displayed as the message in the dialog box. The maximum length of ***prompt*** is approximately 1024 characters, depending on the width of the characters used. If ***prompt*** consists of more than one line, you can separate the lines using a carriage return character (**Chr(**13**)**), a linefeed character (**Chr(**10**)**), or carriage return – linefeed character combination (**Chr(**13**)** & **Chr(**10**)**) between each line.
buttons	Optional. Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If omitted, the default value for ***buttons*** is 0.
title	Optional. String expression displayed in the title bar of the dialog box. If you omit ***title***, the application name is placed in the title bar.
helpfile	Optional. String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If ***helpfile*** is provided, ***context*** must also be provided.
context	Optional. Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If ***context*** is provided, ***helpfile*** must also be provided.

Settings

The ***buttons*** argument settings are:

Constant	*Value*	*Description*
vbOKOnly	0	Display **OK** button only.
vbOKCancel	1	Display **OK** and **Cancel** buttons.
vbAbortRetryIgnore	2	Display **Abort**, **Retry**, and **Ignore** buttons.
vbYesNoCancel	3	Display **Yes**, **No**, and **Cancel** buttons.
vbYesNo	4	Display **Yes** and **No** buttons.
vbRetryCancel	5	Display **Retry** and **Cancel** buttons.
vbCritical	16	Display **Critical Message** icon.
vbQuestion	32	Display **Warning Query** icon.
vbExclamation	48	Display **Warning Message** icon.
vbInformation	64	Display **Information Message** icon.
vbDefaultButton1	0	First button is default.
vbDefaultButton2	256	Second button is default.
vbDefaultButton3	512	Third button is default.
vbDefaultButton4	768	Fourth button is default.
vbApplicationModal	0	Application modal; the user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal; all applications are suspended until the user responds to the message box.
vbMsgBoxHelpButton	16384	Adds Help button to the message box
vbMsgBoxSetForeground	65536	Specifies the message box window as the foreground window
vbMsgBoxRight	524288	Text is right aligned
vbMsgBoxRtlReading	1048576	Specifies text should appear as right-to-left reading on Hebrew and Arabic systems

The first group of values (0-5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512) determines which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the ***buttons*** argument, use only one number from each group.

Note These constants are specified by Visual Basic for Applications. As a result, the names can be used anywhere in your code in place of the actual values.

Return Values

Constant	**Value**	**Description**
vbOK	1	**OK**
vbCancel	2	**Cancel**
vbAbort	3	**Abort**
vbRetry	4	**Retry**
vbIgnore	5	**Ignore**
vbYes	6	**Yes**
vbNo	7	**No**

Remarks

When both ***helpfile*** and ***context*** are provided, the user can press F1 to view the Help topic corresponding to the **context**. Some host applications, for example, Microsoft Excel, also automatically add a **Help** button to the dialog box.

If the dialog box displays a **Cancel** button, pressing the ESC key has the same effect as clicking **Cancel**. If the dialog box contains a **Help** button, context-sensitive Help is provided for the dialog box. However, no value is returned until one of the other buttons is clicked.

Note: To specify more than the first named argument, you must use **MsgBox** in an expression. To omit some positional arguments, you must include the corresponding comma delimiter.