3.1 Introduction

In Topic 2, you learned the basics of VB console programming – how to design and create a simple program, the need for variables, and how to input and output data.

In this topic, we will reinforce these ideas and take a close look at how to manipulate and process both numbers and strings.

3.2 What you should already know

This topic assumes that you have already used and understood the following VB constructs:

- Input boxes and message boxes
- Declaring variables (single, string and integer)
- Console.ReadLine() and Console.Write()

3.3 Learning Outcomes

After completing this topic, you should be able to:

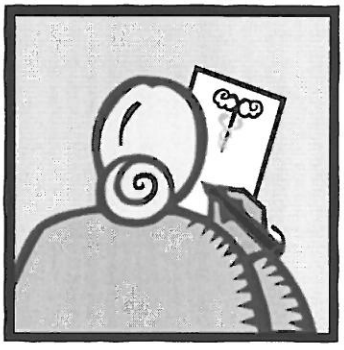
- Design, implement, test and evaluate a VB program
- Save and print a VB program
- Implement calculations involving simple arithmetic
- Use Console.ReadLine(), Console.Write() and Console.Clear()
- Use INT, CINT, ROUND, SQRT and SIN pre-defined numeric functions
- Use LEN, UCase, LCase, Asc, Chr, Mid$ pre-defined string functions
- Carry out systematic and comprehensive testing

3.4 Working with Numbers – example

We are going to consolidate your learning from Topic 2 with a program which carries out some simple calculations.

The example is called 'Bharati's Slab Calculator'!

Here is the problem:



Bharati works in a garden centre, selling paving stones. Customers come in with the plans for their patio and ask how many slabs they will need, and how much it will cost. For example, Mr de Silva says his back garden is 35 slabs wide and 16 slabs deep. He wants the pink granite slabs at £2.99 each. How many slabs will he need, and what will they cost?

The first step is to be absolutely clear about what the program must do. This must be agreed between the customer (Bharati) and the programmer before starting. The agreed definition of what the program must do is called the **program specification**.

 © ZigZag Education 2010

Stage 1 – Analysis: Program Specification

Design, write and test a program to:

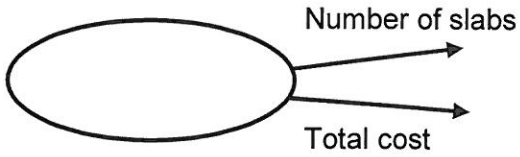
- Input two whole numbers (the number of slabs wide and number of slabs deep)
- Multiply them together (number of slabs needed = number wide x number deep)
- Input the price of a single slab
- Multiply to get the total price
- Display the results (number of slabs required and total cost)

The program should work for any numbers.

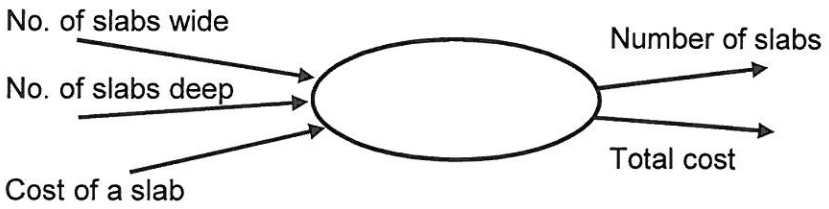
Stage 1 – Analysis – Data Flow Diagram

A 'blob' for the program...

What information comes out of the program?

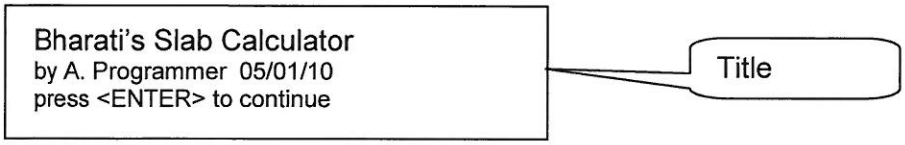


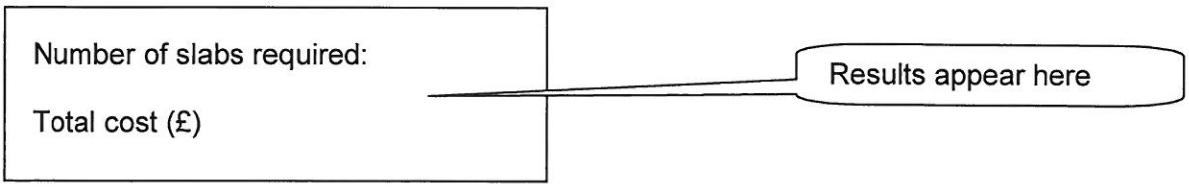
What information does the program need as input?



Stage 2 – Design – User Interface

Sketch out how we want the program to look. We'll have a title window, then use Input Boxes to prompt the user for the number and cost of the slabs, then display the outputs in a new window:





 © ZigZag Education 2010

Stage 2 – Design: Pseudocode

Here is a list of steps (pseudocode) for what this program has to do:

1. Display title window
2. Prompt for and store the number of slabs wide
3. Prompt for and store the number of slabs deep
4. Prompt for and store the cost of one slab
5. Calculate the number of slabs required
6. Calculate the total cost
7. Display the number of slabs required
8. Display the total cost

At the implementation stage, we will turn each of these steps into VB code

Stage 3 – Implementation: Coding

Start a new Visual BASIC Console Application and give it a suitable name.

The code window should already have the first line of code: **Sub Main ()**

Enter the comment lines, as you did in the previous example:

```
' code for Bharatis's Slab Calculator
' by A. Programmer  05/01/10
```

Your name and **today's** date here!

The next stage is to declare all the variables that the program will need.

It will need to store:

- Number of slabs wide (a whole number)
- Number of slabs deep (a whole number)
- The cost of a single slab (a real number, as it could be 2.99)
- The total number of slabs (a whole number)
- The total cost (a real number)

So the next two lines of code will be:

```
Dim wide, deep, no_of_slabs as Integer
Dim slab_cost, total_cost as Single
```

The next three lines of code display the title window, then clear this when the user presses <ENTER>:

```
Console.WriteLine("Bharati's Slab Calculator")
Console.WriteLine("by A. Programmer, 05/01/10")
Console.WriteLIne("press <ENTER> to continue")
Console.ReadLine()
Console.Clear()
```

 © ZigZag Education 2010

The next lines of code are translated from the pseudocode, as shown below:

Pseudocode		Visual BASIC
Prompt for and store the number of slabs wide	→	wide = InputBox("How many slabs wide? ")
Prompt for and store the number of slabs deep	→	deep = InputBox("How many slabs deep? ")
Prompt for and store the cost of one slab	→	slab_cost = InputBox("Cost of one slab? ")
Calculate the number of slabs required	→	no_of_slabs = wide * deep
Calculate the total cost	→	total_cost = no_of_slabs * slab_cost
Display the number of slabs required	→	Console.WriteLine(no_of_slabs & " slabs required")
Display the total cost	→	Console.WriteLine("Total cost will be " & Format(total_cost, "currency"))
(To prevent the results disappearing until <enter> is pressed)	→	Console.ReadLine()

Run the program to make sure it is working correctly.

Enter the following data:
Slabs wide: **4**
Slabs deep: **5**
Cost per slab: **2.99**

The following results should appear in a cleared console window:
20 slabs required
Total cost will be £59.80

If it has worked correctly, save it.

If it doesn't work, then go back and check for errors.

> The most common mistake is to make a spelling error in the name of a variable, so always check these carefully!

Stage 4 – Testing

Testing is a very important stage in the software development process. Proper testing of a commercially produced program may take as long as the implementation.

We will test this program methodically using normal, boundary and erroneous data.

Normal data is data that you would **expect** to be input to the program.

Boundary data *(sometimes called extreme data)* is data that is on the **limits of acceptability** – it should work, but you need to check to make sure. Extreme data could include zero or very large numbers, or numbers close to any limit relevant to the program.

Erroneous data is data that **shouldn't be input under normal use** – for example, entering a letter when asked for a number, or pressing <Enter> when no data has been entered.

 © ZigZag Education 2010

It is best to draw up a table of testing, choosing suitable test data, as shown below.

Fill in the expected results column before running the tests (it predicts what the program should do).

Finally, run the program using your chosen test data, and compare the actual results with the expected results.

If they agree, all is well. If not, you may need to go back and de-bug the program.

Table of testing for Bharati's Slab Calculator

	Inputs			Expected outputs		Actual outputs		Comment
	Wide	Deep	Cost of slab	Total number	Total cost	Total number	Total cost	
Normal data	4	5	2.00	20	40.00			
	10	20	1.99	200	398.00			
	4.5	5.9	2.99	20	59.80			
Boundary data	10000	9000	2.00	90000000	180000000			
	0	any	any	0	0			
	-5	-4	2.50	20	50.00			

Add some other examples of **normal** and **boundary** data to the table, then test the program to make sure it handles them all correctly.

Finally run some **erroneous data** tests, and note the results (either in the table or as notes below it).

Summarise your test results by completing these 3 sentences:

The program gives the correct result if…
The program gives a wrong answer if… because…
The program cannot give an answer if…

Stage 5 – Evaluation
As usual, write a brief report, answering these questions:

- Does the program fulfil the specification (is it effective)?
- Is the program usable?
- Is the program maintainable?

3.5 Arithmetical Expressions

In section 3.4, the example program carried out two simple multiplications, using the lines of code:

```
no_of_slabs = wide * deep

total_cost = no_of_slabs * slab_cost
```

 © ZigZag Education 2010

All other calculations can be carried out in a similar way. Some of the symbols used are the same as in 'normal' arithmetic, but some are different:

For adding, use	+
Subtraction	-
Multiplication	*
Division	/
Raising to a power	^

For complex calculations involving several operations, multiplication and division take precedence over addition and subtraction. However, where the order of the operators matters, it is safest to use brackets.

Here are some examples in Visual BASIC:

```
total = first + second + third

age = 2004 – birth_year

time_in_australia = time_in_scotland + 12

tax = (salary – 4600) * 0.23

years = months / 12

area_of_circle = 3.14 * (radius ^ 2)

volume_of_sphere = (4 * 3.14 * (radius ^ 3)) / 3
```

Note:

(a) The use of brackets where the order is important
(b) The use of readable variable names

3.6 Working with Numbers – Tasks

Some tasks for you to try.

For each task below, you should:

- Clarify the specification (analysis)
- Draw a data flow diagram (analysis)
- Sketch the user interface (design)
- Write pseudocode for the program (design)
- Implement the coding
- Draw up a table of testing
- Test the program with normal, extreme and exceptional data
- Write brief user and technical guides (documentation)
- Evaluate the program

 © ZigZag Education 2010

Design, write and test a program to calculate the volume of a cylindrical water tank, using the formula: volume = $\pi r^2 h$ (r = radius of tank, h = height of tank)

1. Design, write and test a program to calculate the number of points gained by a football team, given the number of wins, draws and lost games, assuming a win is worth 3 points, a draw 1 point, and no points for a lost game

2. Design, write and test a program to calculate the storage requirements in megabytes for bit-mapped graphics. The inputs should be the breadth and height of the graphic in inches, the resolution in dots per inch and the colour depth in bits per pixel.

3.7 Pre-defined numeric functions

There are some standard mathematical calculations that you may want to use in your programs. Visual BASIC (along with most other high level languages) provides pre-defined functions to carry these out for you. We'll take a look at some pre-defined functions provided by Visual BASIC:

INT takes a number and removes any fractional part, leaving the whole number part

CINT takes a number and returns the nearest whole number

ROUND rounds a number to any number of decimal places

SQR returns the square root of any number

SIN returns the sine of an angle (in radians) – multiply the angle by 3.14128 and divide by 180 if you want it to work for degrees

Function Tester Program

We will use a simple VB program to test these functions.

- Start a new VB Console Application
- Copy this code:
- Save the project

```
Sub Main()
  Dim number, result As Single

  Console.Write("Enter a number ...")
  number = Console.ReadLine()

  result = INT(number)

  Console.Write("result is " & result)
  Console.ReadLine()
End Sub
```

Run the program to test the INT function.
As you do so, copy and complete a table of testing as shown below:

Testing the INT function

Input	Expected output	Actual output	Comment
2.5	2		
9.999	9		
9.001	9		
- 5.5	-5		
- 9.001	-9		

 © ZigZag Education 2010

Testing the CINT function

Edit the coding of the program, to change:

result = INT(number) into **result = CINT(number)**

Run the program again and complete a similar table of testing to the one above.

Can you summarise the difference between INT and CINT?

There are many **specialised mathematical functions** available in VB.
You can try some of them next ...

Testing the SQRT function

Edit the coding of the program, to change:

result = CINT(number) into **result = System.Math.Sqrt(number)**

Run the program again and complete a similar table of testing to the one above.

What happens when you enter a negative number? Can you explain this?

What happens when you enter a number like 0.00001? What does this mean?

Testing the SIN function

Edit the coding of the program, to change:

result = SQR(number) into **result = System.Math.Sin(number * 3.14128 / 180)**

(the 3.14128 / 180 part is to make it work for degrees rather than radians)

Run the program again and complete a table of testing like this:

Input	Expected output	Actual output	Comment
0	0		
90	1		
180	0		
30	0.5		
-30	-0.5		

Testing the ROUND function

Edit the coding of the program, to change:

result = CINT(number) into **result = System.Math.Round(number,2)**

Can you summarise what the ROUND function does?

 © ZigZag Education 2010

Other pre-defined functions

Visual BASIC provides many other pre-defined functions, including the other trigonometric functions (cosine and tangent).

It is also possible to create your own functions, so that you are not limited to the pre-defined ones provided by Visual BASIC.

3.8 Working with Strings

A string variable is used to store text, characters and symbols. Each character is stored as a separate extended ASCII 2 byte code. This means that individual characters within a string can be manipulated. VB provides a number of useful pre-defined functions for manipulating strings.

We will use a simple VB program to investigate some of these functions. The first function we will consider is the **Len** function. The **Len** function takes a string as its input, and produces an integer as its output.

String Function Tester Program

- Start a new VB Console application
- Name it "string_function_tester"
- Copy this code: ⟶
- Save the project

```
Sub Main()
    Dim string_in As String
    Dim number_out As Integer

    Console.Write("Enter a string: ")
    string_in = Console.ReadLine()

    number_out = Len(string_in)

    Console.WriteLine("result is " & number_out)
    Console.ReadLine()
End Sub
```

Run the program to test the LEN function.

As you do so, copy and complete a table of testing as shown below:

Testing the LEN function

Input string	Output	Comment
AAAA		
XX XX XX		
Hello, world		
ABC 123 !"£		
10 spaces		

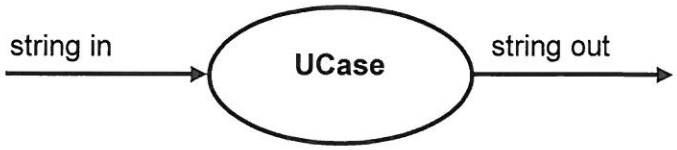
Write a sentence summarising the effect of the **Len** function.

 © ZigZag Education 2010

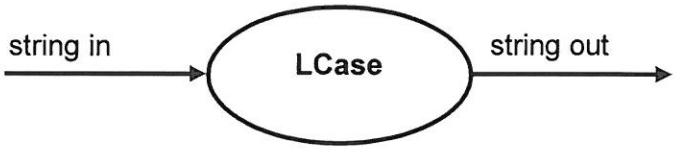
3.9 Other String Functions

As well as **Len**, other pre-defined functions to investigate include:

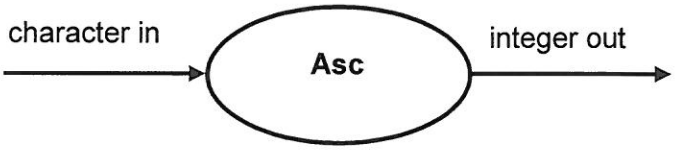
UCase



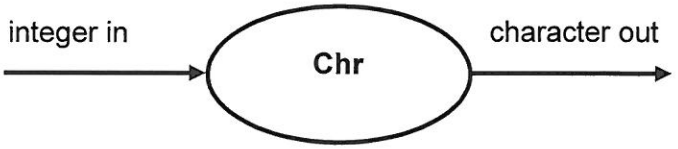
LCase



Asc



Chr



Investigating string handling functions

Adapt the Len function tester program to investigate (in turn) each of the four functions listed above. You will need to make small changes to the code – the first one has been done for you as an example:

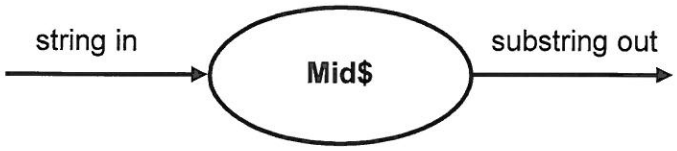
```
Sub Main()
    Dim string_in As String
    Dim string_out As String

    Console.Write("Enter a string: ")
    string_in = Console.ReadLine()

    string_out = Ucase(string_in)

    Console.WriteLine("result is " & string_out)
    Console.ReadLine()
End Sub
```

For each function, draw up a table of testing, and write a sentence summarising the effect of the function.

 © ZigZag Education 2010

3.10 The Mid$ function

Another very useful function is called Mid$. The purpose of Mid$ is to extract a **substring** from a string:



Testing the Mid$ function

Edit the coding of the Function Tester program
to include the line: **string_out = Mid$(string_in, 1, 1)**

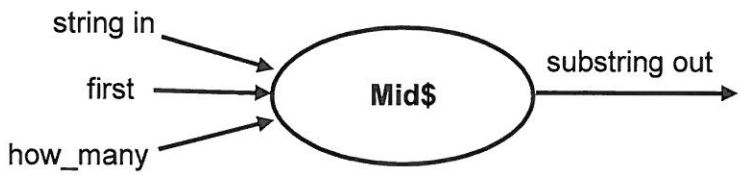
Run the program to test the Mid$ function.
This time you will need to edit the Mid$ line of code each time you run the program.

Record the results in a table, like this, adding a few tests of your own until you are sure you understand how Mid$ works:

Mid$ pre-defined function			
Input string	**Coding used**	**Output string**	**Comment**
Hello, world	Mid$(string_in, 1, 1)		
Hello, world	Mid$(string_in, 2, 1)		
Hello, world	Mid$(string_in, 3, 1)		
Hello, world	Mid$(string_in, 1, 2)		
Hello, world	Mid$(string_in, 1, 3)		
Hello, world	Mid$(string_in, 5, 4)		

Write a brief statement summarising the effect of the **Mid$** function.

It is a bit frustrating having to edit the line of code each time you run the program. You can avoid this by using two integer variables, which we might call **first** and **how_many**. The data flow diagram for the Mid$ function can now be drawn as:



Note: *A function can have any number of input parameters (in this case three) but a maximum of one output parameter. You should find that all the functions you have used fit this pattern.*

 © ZigZag Education 2010

Testing the Mid$ function (2)

Adapt the code of the function tester to prompt the user to enter values for 'first' and 'how_many'.

One possible solution is:

```
Sub Main()
    Dim string_in, string_out As String
    Dim first, how_many As Integer

    Console.Write("Enter a string: ")
    string_in = Console.ReadLine()
    Console.Write("Enter a value for first: ")
    first = Console.ReadLine()
    Console.Write("Enter a value for how_many: ")
    how_many = Console.ReadLine()

    string_out = Mid$(string_in, first, how_many)

    Console.Write("result is " & string_out)
    Console.ReadLine()
End Sub
```

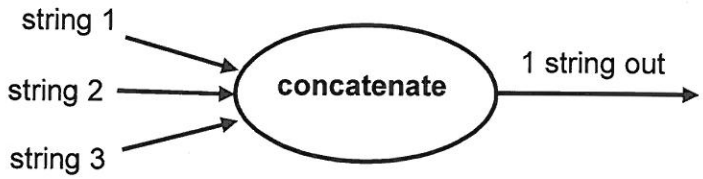
Run the program several times to test it systematically and comprehensively.

The Mid$ function is very useful in many applications, and we will use it later, but first we need to investigate string concatenation.

3.11 String Concatenation

Concatenation is a scary-looking word, but it simply means 'joining together'.

String concatenation means joining two (or more) strings together to form one new string:



String concatenation in VB is very simple – you don't need a special function – you just use the + sign.

For example: if string1 = "Fred",
and string2 = "McSporran",
then string1 + string2 = "FredMcSporran".

 © ZigZag Education 2010

3.12 Practical Task – Devising User IDs

Stage 1 – Analysis: Program Specification

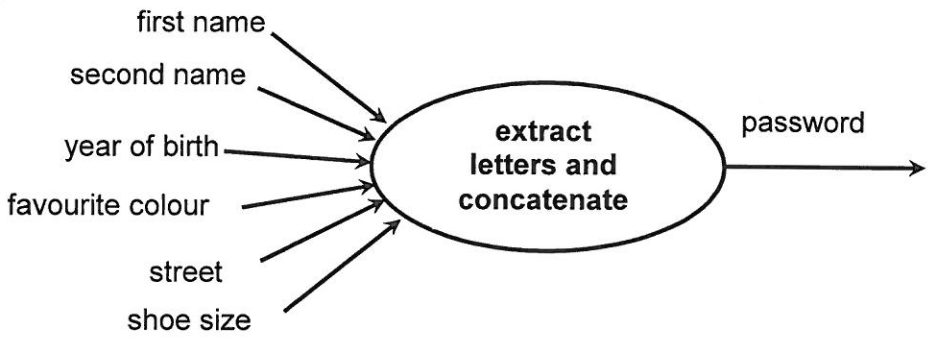
A program is required which will automatically generate a password from information provided by the user. The program should generate the password from the first letter of their first name, second letter of their surname, last two digits of their year of birth, second and third letters of their favourite colour, first three letters of their street name, their shoe size, and the letters VB.

For example:

- **B**renda Mc**S**porran
- Born in 19**87**
- Likes the colour g**ree**n
- Lives in **Mar**ket Place
- Wears size **38** shoes

so her password will be **Bc87reMar38VB**.

Stage 1 – Analysis: Data Flow Diagram



Stage 2 – Design: User interface

We will use input boxes to get the user's input, and a message box to display the password. This is so that the confidential information will not appear permanently on the screen.

Stage 2 – Design: Pseudocode

There are several ways to design the coding for this program. The way we will choose makes the program very readable (and so easily maintained), but it sacrifices efficiency as a result.

The program must carry out the following steps:

1. prompt for the first name
2. extract and store the first character
3. prompt for the second name
4. extract and store the 2nd character
5. prompt for the year of birth
6. extract and store the last 2 digits
7. prompt for the favourite colour
8. extract and store the 2nd and 3rd characters
9. prompt for the street name
10. extract and store the first three characters
11. prompt for and store the shoe size
12. construct and store the password by concatenating the parts
13. display the password in a message box

 © ZigZag Education 2010

Stage 3 – Implementation

Start a new Console Application called Password_generator

At first sight, it looks like the program will need:

⇒ Some string variables (to store the names, colour and street

⇒ Some integer variables (to store the year of birth and shoe size).

However, the numbers (year and shoe size) will **not** be used for calculations, so they all the inputs can be stored as strings.

The variables needed will therefore be:

6 **string** variables to hold the parts: name1, name2, year, colour, street, shoesize

1 **string** variable to hold the password: password

Copy and complete the code for the application, by using Dim to declare these 7 variables, and converting each step of the pseudocode into VB code:

```
Sub Main()
    'coding for the Password program
    'by A. Programmer on 06/01/10

    Dim name1 As String
    ...
    ...
    Console.WriteLine("Password generator")
    Console.Write("Press <Enter> to begin")
    Console.ReadLine()

    name1 = InputBox("Enter your first name")
    name1 = Mid$(name1, 1, 1)

      and so on ...

    password = name1 + name2 + year + colour + street + shoe_size

    MsgBox ("Your password is " & password)
End Sub
```

Note: we are using the same variable (name1) to store the name as input by the user, then overwriting it with the 1st character

Save the project.

Stage 4 – Testing

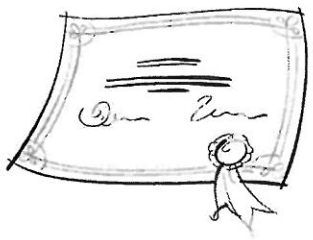
Run some tests using normal, boundary and erroneous data.
Record your results in a table.

Stage 5 – Evaluation

Write a brief evaluation of the program.

 © ZigZag Education 2010

Topic 3 Summary

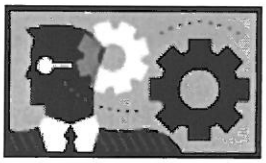


Well done! You have completed Topic 3.

In this topic, you have learned (or revised) how to:

- ☑ Design, implement, test and evaluate a VB program
- ☑ Save and print a VB program
- ☑ Implement calculations involving simple arithmetic
- ☑ Use Console.Clear() to clear the console window
- ☑ Use INT, CINT, ROUND, SQRT and SIN pre-defined numeric functions
- ☑ Use LEN, UCase, LCase, Asc, Chr$, Mid$ pre-defined string functions

Check all the items on this list. If you are not sure, look back through this section to remind yourself.



When you are sure you understand all of these items you are ready to continue with the next topic.

 © ZigZag Education 2010