5.1 Introduction

In Topics 1–4, you have learned how to use a wide range of useful programming constructs. In this topic, you will discover how to make a program repeat sections of code automatically.

5.2 What you should already know

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

- input boxes, message boxes
- variables
- pre-defined functions
- If... Then... Else
- Select... Case

5.3 Learning Outcomes

After completing this topic, you should be able to:

- implement fixed loops using For... Next
- use a counter in a fixed loop
- implement conditional loops using Do... Loop Until
- use Random numbers
- implement nested loops
- compare Do... Until and Do... While loops

5.4 Repetition using a fixed loop

So far, every program you have written starts at the beginning, executes each line once, then stops at the end. If you want to repeat the program you have to click on the start icon to run it again. It is often useful in a program to be able to repeat a line or group of lines automatically.

To do this, you can use a **FOR... NEXT loop**.

Here is a simple example program that would benefit from a FOR... NEXT loop.

Enter this coding:

```
Sub Main()
  ' code to display 10 greetings on the screen
  ‘ by a not very good programmer
  ‘ who hasn't been taught about FOR... NEXT loops

  Console.WriteLine("Have a nice day!")
  Console.WriteLine("Have a nice day!")
  Console.WriteLine("Have a nice day!")
  Console.WriteLine("Have a nice day!")
  Console.WriteLine("Have a nice day!")
  Console.WriteLine("Have a nice day!")
  Console.WriteLine("Have a nice day!")
  Console.WriteLine("Have a nice day!")
  Console.WriteLine("Have a nice day!")
  Console.WriteLine("Have a nice day!")

  Console.ReadLine()
End Sub
```

Here is a second version of the program that uses a FOR... NEXT loop to cut down the amount of coding required:

```
Sub Main()
  ' code to display 10 greetings on the screen
  ‘ by a much better programmer
  ‘ using a FOR... NEXT loop

  Dim counter As Integer

  For counter = 1 To 10
     Console.WriteLine("Have a nice day!")
  Next

  Console.ReadLine()
End Sub
```

Alter the coding as shown and run the program again.

It does exactly the same, but takes much less coding.

A For .. Next loop is known as a "fixed loop" because the programmer decides how many times the code is to be repeated.

5.5 Repetition using For... Next

In this example, we will develop a program to display a repeated message on screen.

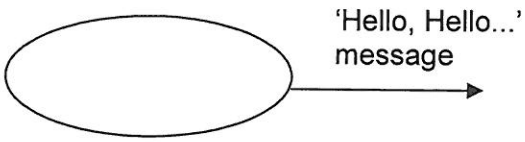
Stage 1 – Analysis

Program Specification

Design, write and test a program to display the message

'Hello, Hello, Hello ...(25 times)'.

Data Flow Diagram



Stage 2 – Design: User Interface

The output will be displayed in a cleared console window.

Stage 2 – Design: Pseudocode

Next, we design the list of steps (pseudocode) and then the coding for the command button:

Pseudocode	Visual BASIC coding
1. Do the following 25 times 2. Add the word 'Hello' to the list box	For Counter = 1 to 25 Console.WriteLine("Hello") Next

*Note that it is usual to **indent** the code within the loop to improve readability*

Stage 3 – Implementation

- Start a new Visual BASIC Console Application project
- Enter the code for the program (declare a variable Counter as Integer)

Stage 4 – Testing

Run the program to make sure it works correctly.
There is no need for a table of testing for a simple program like this.

Modifications (1)

Alter the coding so that it displays the message

(1) 'Goodbye' 12 times
(2) 'I must work harder' 200 times
(3) 'This is very easy' 100 times

Modifications (2)

The program would be much more useful if it was possible to make changes to the message and the number of times it was displayed, without having to alter the coding each time. This can be achieved by using **variables**.

Change the coding as follows:

```
Sub Main()
 ' improved For... Next example

 Dim counter As Integer
 Dim message As String
 Dim how_many As Integer

 message = InputBox("Message required...")
 how_many = InputBox("How many repetitions?")

 For counter = 1 To how_many
  Console.WriteLine (message)
 Next

Console.ReadLine()
End Sub
```

A string variable called **message** will store the message, and an integer variable called **how_many** will store the number of repetitions

Instead of a fixed number here, the loop will continue up to the number stored in **how_many**

Whatever string is stored in the variable called **message** will be displayed in the window

Testing (continued)

Now test the program thoroughly using **normal**, **boundary** and **erroneous** values for both 'message' and 'how_many'.

Stage 5 – Evaluation

As usual, you should:

- print out hard copies of the coding
- save your program
- write a brief evaluation of the program

Modifications (3)

At the moment, this program can display any message over and over again, but it is the same message on each line.

Can you adapt the program to produce displays like:

Tick Tock Tick Tock Tick Tock Tick Tock Tick

Left Right Left Right Left Right Left Right Left

Go home Now! Go home Now! Go home Now!

Hint: you will need 2 (or 3) lines of code within the For... Next loop.

5.6 Counting using For... Next

1,2,3 ...

The standard For... Next loop that we have used so far is of the format

For counter = 1 to maximum
Action
Next

Note that we have called the loop variable 'counter' (because that is what it does), but it can be called anything you like. The following versions would work in exactly the same way:

For silly_name_for_a_variable = 1 to maximum
Action
Next

For i = 1 to maximum
Action
Next

The last of these (using i as the loop variable) is very common, but counter is a good, readable variable name.

For... Next loops are an example of '**fixed loops**'. This is because the number of times the action is executed is fixed in advance by the programmer, using the value of maximum.

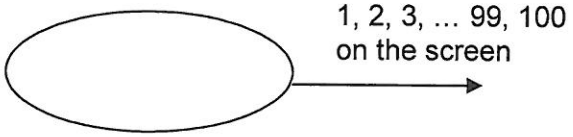
Later, we will see that it is possible to construct loops where the number of times the action is executed is NOT known in advance.

Example 1 – Simple Counting Program

Stage 1 – Analysis: Program Specification

Design, write and test a program to display 1, 2, 3, 4, 5... 99, 100.

Stage 1 – Analysis: Data Flow Diagram



Stage 2 – Design

We want the numbers displayed in a column in a cleared window.

Here is the list of steps (pseudocode) and then the coding for the command button:

Pseudocode	Visual BASIC coding
1. Do the following 100 times 2. Display the **counter** in the window	For counter = 1 to 100 Console.WriteLine(counter) Next

Instead of a "message" being displayed, the current value of counter is displayed instead

Stage 3 – Implementation

Start a new Visual BASIC console application project
Enter the program code (declare a variable Counter as integer)

Stage 4 – Testing

Run the program to make sure it works correctly (it should produce a list of numbers from 1 to 100 in the window).

There is no need for a table of testing for a simple program like this.

Example 2 – Experimenting with the Simple Counting Program

You are going to use the simple counting program as a template to experiment with For... Next loops.

In each case below:

- replace the line of code **For counter = 1 To 100** with the modification suggested
- run the program
- note the results in a table like this:

Coding used	Results
For counter = 1 To 100	1 2 3 4... 99 100

Modification (1) For counter = 1 To 9999

Modification (2) For counter = 1 To 100 Step 2

Modification (3) For counter = 2 To 100 Step 2

Modification (4) For counter = 0 To 100 Step 10

Modification (5) For counter = -10 To 10 Step 5

Modification (6) For counter = 100 To 1 Step –5

Modification (7) For counter = 0 To 5 Step 0.5

*The final modification will require a change to another line of coding. Can you work out what it will be? Hint: 0.5 is **not** an integer*

Write the Visual BASIC coding of a For... Next loop to produce each of the following lists of numbers:

(a) 3, 6, 9, 12, 15, 18 ..... 33, 36

(b) 0, 9, 18, 27 99

(c) 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

(d) 0, 0.75, 1.5, 2.25, 3, 3.75, 4.5

(e) 50, 40, 30, 20, 10, 0, -10, -20, -30, -40, -50

(f) 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 *(hint: these are all numbers **squared**)*

(g) 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 *(hint: these are **powers of 2**)*

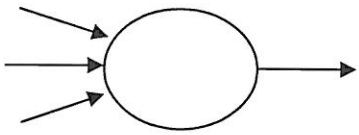
Example 3 – General Purpose Counting Program

Stage 1 – Analysis

Program Specification

Design, write and test a program to display any list of numbers, given the starting number (lower limit), the final number (upper limit) and the step size.

Data Flow Diagram
(copy and complete...)



Stage 2 – Design

You will use input boxes to enter the lower limit, upper limit and step sizes.

Here is a possible list of steps (pseudocode):

Copy and complete the coding yourself

Pseudocode	Visual BASIC coding
1. Set up variables to store the 3 numbers and the loop counter	Dim ... Dim ... Dim ... Dim ...
2. Store the lower limit entered by the user	lower = InputBox(.....)
3. Store the upper limit entered by the user	upper = ...
4. Store the step size entered by the user	stepsize = ...
5. Repeat the following, starting at lower limit, and going up to upper limit in steps of stepsize	For ... = ...To ... stepsize ...
5.1. Display the counter	Console.Writeline (...)
6. End of loop	Next

Stage 3 – Implementation

- Start a new Visual BASIC console application project
- Enter the program code

Stage 4 – Testing

Carry out systematic testing of the program, completing a table like this:

	Inputs			Expected outputs	Actual outputs	Comment
	Lower limit	Upper limit	Step size			
Normal data	10	20	3	10, 13, 16, 19		
	1000	8000	2500	1000, 1250, 1500, 1750		
	10	0	-2	10, 8, 6, 4, 2, 0		

Devise your own test data, covering a range of normal, boundary and erroneous data.

Write a short summary of your testing.

If all the test results were as expected, move on to stage 5. If not, go back and correct your coding until it works correctly.

Stages 5 – Evaluation

As usual, you should

- print out a hard copy of the coding
- save your program
- write a brief evaluation of the program

Example 4 – Multiplication Tables

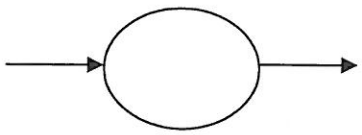
This program uses a For .. Next loop, with its loop counter, to do something useful!

Stage 1 – Analysis

Program Specification

Design, write and test a program to display any multiplication table (chosen by the user) in the form: 1 x 5 = 5, 2 x 5 = 10 and so on as far as 12 x 5 = 60

Data Flow Diagram
(copy and complete ...)



Stage 2 – Design

The program should use an input box to get the user's choice of table, and display the output in a window, like this:

```
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
....
```

Each line that appears in the output window should look like this (made up of 5 parts):	2	X	5	=	10
	the counter (1,2,3 ...)	*the symbol X*	*the multiplier chosen by the user*	*the symbol =*	*the answer (counter x multiplier)*

To do this, we need several variables:

- the counter (an integer)
- the multiplier (an integer supplied by the user)
- the answer (an integer calculated by the program)
- a string variable, which we'll call 'message', which stores together the 5 parts shown above into a single message to display in each line of the list box

The line of code to assemble this message will look like:

```
message = counter & " x " & multiplier & " = " & answer
```

Here is the list of steps (pseudocode):
Copy and complete the coding yourself

Pseudocode	Visual BASIC coding
1. Set up variables to store the 3 numbers and the message	Dim ... Dim ... Dim ... Dim ...
2. Store the multiplier entered by the user	multiplier = Input Box (...)
3. Repeat the following from 1 to 12	For... = ...To... Step...
3.1. calculate the answer	answer = counter * multiplier
3.2. assemble the message	message =...
3.3. display the message	Console.WriteLine(message)
4. Next	Next

Stage 3 – Implementation

- Start a new Visual BASIC console application project
- Enter the program code
- Save the completed project

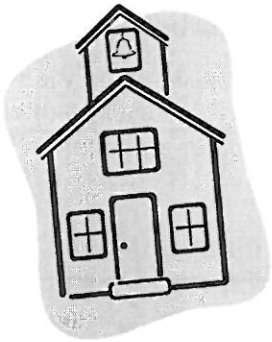
Stages 4 and 5 – Testing and Evaluation

Complete the testing and evaluation of the program in the usual way.

5.7 For... NEXT tasks

Choose one (or more) of the following program specifications, and design, implement and test a program to fulfil the specification. Work through all the stages of the software development process from analysis to evaluation for your chosen task.

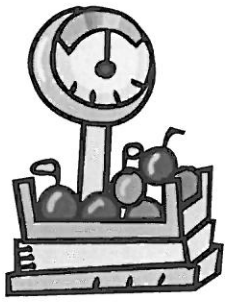
Times Tables (advanced version)



A primary school teacher wants a program which will allow a pupil to type in any whole number. The program will then display the relevant times table, up to a maximum multiplier set by the pupil. The display should be in the format:

5 times 1 equals 5
5 times 2 equals 10
5 times 3 equals 15, and so on ...

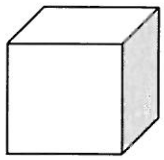
Cost and weight calculator



A greengrocer needs a program which will allow him to type in the price of 1kg of any item. The program should then display the cost of 0, 0.1, 0.2... up to 1.8, 1.9, 2.0 kg of the item.
The output might look something like this:

0 kg costs £0
0.1 kg costs £0.20
0.2 kg costs £0.40
0.3 kg costs £0.60, and so on ...

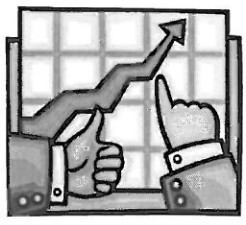
Cubic numbers



A mathematician wants a list of cubic numbers (1, 8, 27, 64, 125 ....) starting and finishing at any point on the list. The results should be displayed like this:

2 cubed = 8
3 cubed = 27
4 cubed = 64, and so on....

Quadratic function calculator



A pupil has been asked to draw a graph of the function $y = 3x^2 + 4$.
She needs a table of the values of the function between –5 and +5.
She is not sure about the step size between points, so wants the program to allow her to choose any step size.

The results should be displayed like this:

x = 1 >>>>>>>> y = 7
x = 2 >>>>>>>> y = 16
x = 3 >>>>>>>> y = 31 and so on...

5.8 Do... Loop Until

We have used **For .. Next loops** to repeat a section of program a fixed number of times. This is fine if we know how many times the section of program is to be repeated.

What about times when the number of repetitions is not known in advance?

For example, a quiz program might give the user repeated chances to get the answer correct. The programmer doesn't know in advance whether the user will get the question right first time, or take 2, 3, 4 or more attempts.

In this type of situation, the programmer needs to use another kind of loop. Visual BASIC provides several other types of loop. We will use a type called **Do... Loop Until.**

The pattern (syntax) for this type of loop is very simple:

Do
line(s) of code
to be repeated
Loop Until *condition*

5.9 Do... Loop Until example program:

Start a new console application.

Enter the following program code.

```
Sub Main()
  ' generates a question to the user
  ' and waits for the correct answer

  Dim user_answer As Integer
  Dim correct_answer As Integer

  correct_answer = 4

  Do
    user_answer = InputBox("What is 2 + 2?")
  Loop Until user_answer = correct_answer

  MsgBox ("Well done!")
End Sub
```

The program should keep repeating the Input Box line of code until the user's answer is the same as the correct answer.

Run the program. Test it with right and wrong answers.

Does it behave as predicted?

Adapt the program to ask

1. a different arithmetical question (e.g. What is 100 x 100?)
2. a general knowledge question (e.g. Who won MasterChef in 2010?)

Improvements to the program

This simple program works fine, but there are some obvious changes which would improve it!

Improvement 1
When you give the wrong answer, the program doesn't tell you. It could be improved by presenting a message which told the user to try again.

To do this, you need to add in the following line of code:

If user_answer <> correct_answer Then MsgBox "Wrong, try again!"

Can you work out where this line of code should go?
Edit it into your program, and check that it works.

Improvement 2
The program would be improved if it told you how many guesses you made before you got the correct answer.

To do this, we need to include a **counter** in the loop.

Here is the pseudocode (the new sections are in bold).

1. *Declare integer variables for the user's answer and the correct answer*
2. ***Declare an integer variable for the counter***
3. ***Set the counter equal to zero***
4. *Set the correct answer equal to 4*
5. *Do*
   5.1. Get the user's answer to the question (What is 2 + 2)
   5.2. Add one to the counter
   5.3. If the answer is not the correct answer, display 'Wrong, try again' message
6. *Until user's answer is equal to the correct answer*
7. ***Display message (Well done! You got that right in ....)***

The code for this is **counter = counter + 1**

Turn the pseudocode into VB, and adapt your program accordingly.

Make sure you save this program as we will use it again in topic 5.11

Edit the changes into your program, and check that it works.

Improvement 3
The 3rd improvement would be if the program could be made to ask a different question each time, instead of always asking 2 + 2. To do this, we need to use VB's random number generator. See topic ***5.10 Random Numbers****.*

5.10 Random Numbers

Visual BASIC provides the programmer with a pre-defined function **Rnd** to generate random numbers. Before we use it in the arithmetic tester program, we will use a simple program with a For... Next loop to learn how the **Rnd** function operates.

Start a new console application project.
Enter the following program coding:

```
Sub Main()
  ' generates lists of random number

  Dim number As Single
  Dim counter As Integer

  For counter = 1 To 10
    number = Rnd
    Console.WriteLine (number)
  Next

  Console.ReadLine()
End Sub
```

Run this program.

Write down the list of 10 random numbers produced.
Note that the **Rnd** function produces random numbers **between 0 and 1**.

We would rather have **random whole numbers between 1 and 10**.

To do this, we need to do 3 things to the line **number = Rnd**.

- First, we need to **multiply by 10** to produce a random real number between 0 and 10
- Then we need to 'chop off' the fraction part using the pre-defined function **Int**
- Finally, we need to **add 1**, otherwise the highest number will always be 9, as it is rounded down by the Int function

Edit the line: **number = Rnd**
To: **number = Int (Rnd * 10) + 1**

Now run the program again. Write down the list of random numbers.
This time they should all be whole numbers between 1 and 10.
Stop the program.

Run the program again.
And again!

You will notice that it always generates the **same** list of 'random' numbers.

To make them really random, you need to add the keyword **Randomize** at the start of the program (anywhere before the keyword **Rnd**).

The coding for your random number generator should now look like this:

```
Sub Main()
   ' generates lists of random number
   Dim number As Integer
   Dim counter As Integer

   Randomize()
   For counter = 1 To 10
      number = Int(Rnd() * 10) + 1
      Console.WriteLine(number)
   Next
   Console.ReadLine()
End Sub
```

You can now declare the number to be an integer, as we are only going to allow whole numbers.

You can adapt this line in a variety of ways to produce other sets of random numbers.

For example, to produce:

random numbers between 1 and 20, change it to **number = Int (Rnd * 20) + 1**

random numbers between 51 and 60, change it to **number = Int (Rnd * 10) + 51**

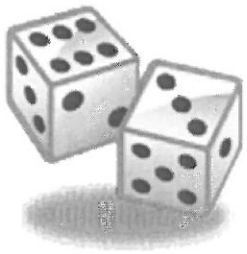
random **even** numbers between 0 and 10, change it to **number = 2 * Int (Rnd * 6)**

Try experimenting with this line until you understand how it works. Sometimes it takes a little thought to work out exactly what numbers to put in, so that you get the right range and don't miss out the highest or lowest number.

Test your program with the following lines of code.
In each case, make a note of the results, and explain what you get.

Line of code	Results	Explanation
number = Int(Rnd * 200) + 1		
number = Int(Rnd * 20) + 1		
number = Int(Rnd * 100) + 2000		
number = 2 * (Int(Rnd * 50) + 1)		
number = 2 ^ (Int(Rnd * 8) + 1)		

Tasks:



- Modify your program so that when you click on the button it produces a single dice roll (a random number between 1 and 6). *Hint: you won't need a For... Next loop.*
- Modify your program so that it produces a double dice roll, and displays the number on each dice **and** the total score. *Hint: you will need to generate 2 random numbers every time you click the button.*

5.11 Arithmetic Tester

We can now combine what we have learned about random numbers with our arithmetic tester program from topic 5.9.

Here is the current version of the program:

```
Sub Main()
   ' generates a question to the user
   ' and waits for the correct answer

   Dim user_answer, correct_answer As Integer
   Dim counter As Integer

   counter = 0
   correct_answer = 4

   Do
      user_answer = InputBox("What is 2 + 2?")
      counter = counter + 1
      If user_answer <> correct_answer Then MsgBox("Wrong, try again!")
   Loop Until user_answer = correct_answer

   MsgBox("Well done! You got that right in " & counter)
End Sub
```

- Open this project
- Modify the coding as below (changes in bold):

```
Sub Main ()
 ' generates a random question to the user
 ' and waits for the correct answer

 Dim user_answer, correct_answer As Integer
 Dim first, second as Integer
 Dim counter as Integer

 Randomize
 first = Int(Rnd * 10) + 1
 second = Int(Rnd * 10) + 1
 counter = 0
 correct_answer = first + second

 Do
  user_answer = InputBox("What is " & first & " + " & second & "?")
  counter = counter + 1
  If user_answer <> correct_answer Then MsgBox("Wrong, try again!")
 Loop Until user_answer = correct_answer
 MsgBox("Well done!  You took " & counter & " tries")
End Sub
```

The variables first and second hold the 2 random numbers to be used for the question.

These lines generate the 2 random numbers for the question

This displays the value of the variables first and second, rather than the numbers 2 + 2

One more modification!

The program only asks **one** random additional question each time it is run.

By adding 3 lines of code, we can make it give the user a series of (say) 6 questions.
We can do this by putting the whole of the middle section of the program inside a For... Next loop, like this:

```
Sub Main ()
 ' generates 6 random question to the user
 ' and waits for the correct answer

 Dim user_answer As Integer
 Dim correct_answer As Integer
 Dim first as Integer
 Dim second as Integer
 Dim counter as Integer
 Dim question as Integer

 Randomize
 For question = 1 To 6
   first = Int(Rnd * 10) + 1
   second = Int(Rnd * 10) + 1
   counter = 0
   correct_answer = first + second

   Do
     user_answer = InputBox("What is " & first & " + " & second & "?", "Question" & question)
     counter = counter + 1
     If user_answer <> correct_answer Then MsgBox("Wrong, try again!")
   Loop Until user_answer = correct_answer
   MsgBox("Well done!  You took " & counter & " tries")
 Next
End Sub
```

This new variable here is the loop counter for the For... Next loop.

Here is the For... Next loop to repeat the next section 6 times.

This adds the Question number as a title to the Input Box.

You now have a loop within a loop. The technical term for this is **nested loops.**

Make the above changes to the coding.

Save the revised project.

Carry out some thorough testing of your program, using normal, boundary and erroneous data.

Modify the program so that it:

- asks multiplication questions rather than addition
- uses random numbers between 1 and 12
- asks 5 questions
- displays 'Well done – right first time!' if the user gets it right first time
- displays 'Keep practising! You took X tries to get it right!' otherwise

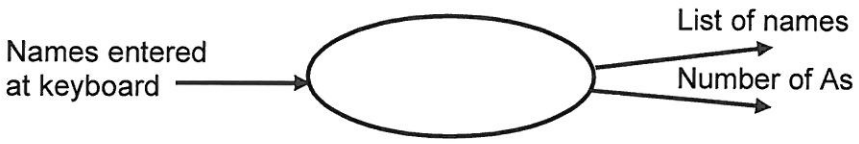
Test the program using normal, boundary and erroneous data.
Write an evaluation report.

5.12 More examples using Do... Loop Until

Example 1 – Class lists

Design, write and test a program for a teacher. The program should prompt the user to enter any list of names, which will be displayed on the screen. The program should count how many of these names begins with the letter A, and display this information at the end of the list.

Stage 1 – Analysis: Data Flow Diagram



Stage 2 – Design

The program will use input boxes to prompt for names to be entered.

Next, we design the list of steps (pseudocode)
and then the coding for the button's click event:

Q: *We will need to use a loop. Should it be a For... Next loop, or a Do... Loop Until?*

A: As we don't know in advance how many names there will be in the list, we need to use a **Do... Loop Until**.

Q: *What condition will we use to stop the loop?*

A: Ask the user to enter the word END after entering all the names. The loop can then continue **until name = "END"**

Pseudocode	Visual BASIC coding
1. Set a counter equal to zero	counter = 0
2. Do the following:	Do
2.1 Prompt the user to enter a name	name = InputBox("Enter a name (or END)")
2.2 Add the name to the list	Console.WriteLine(name)
2.3 Extract the first letter of the name	initial = Mid$(name,1,1)
2.4 If the first letter is A, add 1 to the counter	If initial = "A" Then counter = counter + 1
2.5 Until the user enters 'END'	Loop Until name = "END"
3. Display the counter at the end of the list	Console.WriteLine(counter)

Variables required:
counter (integer), name (string), initial (string)

Stage 3 – Implementation

- Start a new Visual BASIC console application project
- Enter the code for the program (declare all required variables)
- Save the project

Stage 4 – Testing

Test the program with the following sets of test data, and add some more of your own.

	Test data 1	Test data 2	Test data 3
Test data	Andrew Bill Cliff Doris Sarah END	Alison Albert Bill Bert Ahmed end	Alison alison Albert ahmed END
Expected answer	1		
Comment	OK, but END should not be shown		

You should have noticed 3 problems with the program:

1. It doesn't count names which start with a **lower case** 'a'
2. It adds the word END to the list
3. It doesn't stop when you enter 'end' in **lower case**

You should be able to modify your code to solve these problems.

Hints:

1. Use the function Ucase in step 3.4
2. Make step 3.2 conditional (If Ucase(name) <> "END" Then .....)
3. Make the end of loop condition into a complex condition using OR

Stage 5 – Evaluation

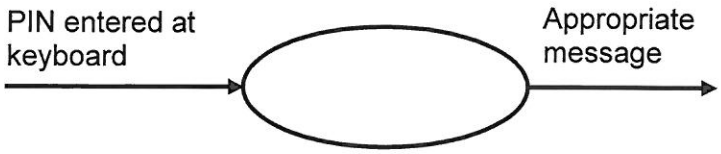
As usual, you should:

- print out hard copies of your coding
- save your completed program
- write a brief evaluation of the program

Example 2 – Password Protection

Design, write and test a program for a bank cash machine. The program should prompt the user to enter their PIN. If the PIN is correct, it should display 'Welcome to the VB Bank' (message 1). If not, it should notify the user that their PIN was entered wrongly (message 2), and let them try again, but only allow 3 tries.
If the user enters their PIN wrongly 3 times, they should be warned that their card is being kept (message 3).

Stage 1 – Analysis: Data Flow Diagram



Stage 2 – Design

The user will be prompted to enter their PIN through an Input Box. The appropriate message will be displayed to the user using a Message Box.

First, we design the list of steps (pseudocode) and then the coding for the program:

We will use a **Do... Loop Until**, as the number of attempts the user makes is unknown in advance by the programmer.

The condition to end the loop will be that the PIN is correct OR that the user has already had three attempts.

Pseudocode	Visual BASIC coding
1. set a counter equal to zero	counter = 0
2. store correct PIN	correct_pin = 1347
3. do the following:	Do
3.1 prompt the user to enter their PIN	pin = InputBox("Enter your PIN")
3.2 If PIN is correct display message(1) or else display message(2)	If pin = correct_pin then MsgBox("Welcome to VB Bank") Else MsgBox ("PIN entered wrongly – try again")
3.3 add 1 to the counter	counter = counter + 1
4. until the PIN is correct or counter > 3	Loop Until (pin = correct_pin) Or (counter > 3)
5. If counter > 3 then display message(3)	If counter > 3 Then MsgBox("The card is being kept for security")

Variables required:

counter (integer)
pin (integer)
correct_ pin (integer)

Stage 3 – Implementation

- Start a new Visual BASIC console applications project
- Enter the code for the program (declare all required variables)
- Save the project

Stage 4 – Testing

Create some suitable test data, and use it to test the program

	Test data 1	Test data 2	Test data 3
Test data	1347	9999 8888 1347	1234 4321 9999
Comment			

Stages 5 – Evaluation

As usual, you should:

- print out hard copies of your form and the coding
- save your program
- write a brief evaluation of the program

Task: Modifying the program

Modify the program to:

- prompt the user to enter a password (which could contain letters as well as numbers)
- allow 5 attempts at guessing the password

5.13 Other forms of Conditional Loop

There are 4 variations of conditional loop in Visual BASIC.

So far, we have only used the Do... Loop Until form of loop.

In some high level languages, this is the only kind of conditional loop, and it is possible to manage without the other kinds.

However, for completeness, here is a brief summary of all four types.

Type of loop	Syntax	Comments
Do... Loop Until	**Do** *Line(s) of code* *to be repeated* **Loop Until** *condition*	Always executed at least once, as condition is tested at the **end**; stops when the condition becomes **true**
Do Until... Loop	**Do Until** *condition* *Line(s) of code* *to be repeated* **Loop**	Only executed if the condition is true, as it is tested at the **beginning**
Do... Loop While	**Do** *Line(s) of code* *to be repeated* **Loop While** *condition*	Loops **while** the condition is true, and stops when the condition becomes **false**
Do While... Loop	**Do While** *condition* *Line(s) of code* *to be repeated* **Loop**	Only executed **while** the condition is true, as it is tested at the **beginning**

Each type of loop has advantages and disadvantages. You can choose to use Do... Loop Until all the time. However, it is useful to be able to use all four versions.

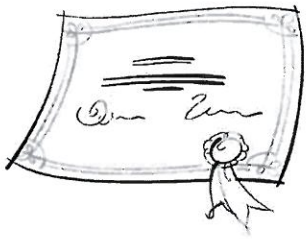
Task: Using different forms of Do... Loop

Modify the simple arithmetic tester program to use Do... Loop While

Question:

Why is the Do Until or Do While loop not suitable for the arithmetic tester program?

Topic 5 Summary



Well done! You have completed Topic 5.

In this topic, you have learned how to:

- ☑ Implement fixed loops using For... Next
- ☑ Use a counter in a fixed loop
- ☑ Implement conditional loops using Do... Loop Until
- ☑ Use Random numbers
- ☑ Implement nested loops
- ☑ Compare Do... Until and Do... While loops

When you are sure you understand all of these items, you have completed this short course in VB console programming.

You should now progress to Topics 6-10, which is based on VB Windows Applications Programming.

Answers to Topic 2 exercises

2.7

(a) integer
(b) string
(c) single
(d) Boolean
(e) single
(f) integer
(g) Boolean
(h) integer
(i) string
(j) single
(k) single
(l) string
(m) string
(n) single

2.10

Format(average, ".000")	Displays average to 3 decimal places
Format(average, "000.0")	3 digits before the point, 1 after
Format(average, "###.0")	3 digits before the point (if required), 1 after
Format(average, "fixed")	Always showing 2 decimal places after the point
Format(average, "currency")	2 decimal places and a £ sign
Format(average, "percent")	Converts a fraction into a percentage (e.g. 0.5 becomes 50%)
Format(average, ".00\s")	Displays 2 decimal places, and adds s at the end
Format(average, "\{00.00\}")	Displays 2 digits before and after the point, and puts {} brackets around the answer

2.12 (Practical Task)

```
Sub Main()
    'distance, speed and time program
    'code for the start command button
    'by A. Programmer on 06/01/10

    Dim start_town, end_town As String
    Dim distance, av_speed As Integer
    Dim time_taken As Single

    start_town = InputBox("Where are you travelling from?")
    end_town = InputBox("Where are you going?")
    distance = InputBox("How far is it (to the nearest km)?")
    av_speed = InputBox("What is your average speed (to the nearest km/hour)")

    time_taken = distance / av_speed

    Console.WriteLine("From: " & start_town & VbCrLf & "To: " & end_town)
    Console.WriteLine(""Distance: " & distance & " km" & VbCrLf & "Av. speed: " & av_speed & "km/hr"
    & VbCrLf & "Time: " & Format(time_taken, "#0.0\hrs"))
    Console.ReadLine()
End Sub
```

Note: *If you need to split a long line like this one, you can insert:* ***<space><underscore><space><return>****; this splits the line into two, but VB continues to treat it as a single line of code.*

Answers to Topic 3 exercises

3.6

Lines of code required for the calculation in each of the programs:

1. volume = 3.14 * (radius ^ 2) * height

2. points = (3 * wins) + draws

3. bits = (breath * res) * (height * res) * bits_per_pixel
   megabytes = bits / (8 * 1024 * 1024)

3.9 (Testing the LEN function)

Input string	Output	Comment
AAAA	4	4 characters
XX XX XX	8	8 characters, including the 2 spaces
Hello, world	12	12 characters, including the space and comma
ABC 123 !"£	11	Counts all characters (including letters, numeric digits, spaces, symbols)
10 spaces	18	2 digits, 10 spaces, 6 characters

3.9 (String function programs – solution)

UCase

```
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(string_out)
```

LCase

```
Dim string_in As String
Dim string_out As String
Console.Write("Enter a string: ")
string_in = Console.ReadLine()
string_out = LCase(string_in)
Console.WriteLine(string_out)
```

Asc

```
Dim character_in As String
Dim number_out As Integer
Console.Write("Enter a character: ")
character_in = Console.ReadLine()
number_out = Asc(character_in)
Console.WriteLine(number_out)
```

Chr

```
Dim number_in As Integer
Dim character_out As String
Console.Write("Enter a number: ")
number_in = Console.ReadLine()
character_out = Chr(number_in)
Console.WriteLine(character_out)
```


3.10

Mid$ pre-defined function			
Input string	**Coding used**	**Output string**	**Comment**
Hello, world	Mid$(string_in, 1, 1)	H	1st character
Hello, world	Mid$(string_in, 2, 1)	e	1 character, starting at 2nd
Hello, world	Mid$(string_in, 3, 1)	l	1 character, starting at 3rd
Hello, world	Mid$(string_in, 1, 2)	He	2 characters, starting at 1st
Hello, world	Mid$(string_in, 1, 3)	Hel	3 characters, starting at 1st
Hello, world	Mid$(string_in, 5, 4)	o, w	4 characters (including the space), starting at 5th

3.12 (Devising user IDs – solution)

```
Sub Main ( )
   'coding for the Password command button
   'by A. Programmer on 19/04/10

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

    Dim name1, name2, year, colour, street, shoe_size, password As String

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

   name2 = InputBox("Enter your surname")
   name2 = Mid$(name2, 2, 1)

   year = InputBox("Enter your birth year (e.g. 1993)")
   year = Mid$(year, 3, 3)

   colour = InputBox("Enter your favourite colour")
   colour = Mid$(colour, 2, 2)

   street = InputBox("Enter the name of the street where you live")
   street = Mid$(street, 1, 3)

   shoe_size = InputBox("Enter your shoe size")

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

   MsgBox("Your password is " & password)

End Sub
```


Answers to Topic 4 exercises

4.9 (Multiple Ifs – solution)

There are many possible solutions. Here is one version:

```
Sub Main ( )
    ' coding for the OK command button
    ' displays an appropriate message for each grade letter entered
    ' and changes the form colour
    ' written by A. Programmer on 19/04/10

    Dim Grade As String

    Console.Write("Enter a grade (A to F)")
    Grade = Console.ReadLine()

    If Grade = "A" Then
       Console.ForegroundColor() = ConsoleColor.Red
       MsgBox(Grade & " means you got over 70%")
    End If

    If Grade = "B" Then
       Console.ForegroundColor() = ConsoleColor.Blue
       MsgBox(Grade & " means you got between 60% and 70%")
    End If

    If Grade = "C" Then
       Console.ForegroundColor() = ConsoleColor.Green
       MsgBox(Grade & " means you got between 50% and 60%")
    End If

    If Grade = "D" Then
       Console.ForegroundColor() = ConsoleColor.Yellow
       MsgBox(Grade & " means you got between 40% and 50%")
    End If

    If Grade = "F" Then
       Console.ForegroundColor() = ConsoleColor.Black
       MsgBox(Grade & " means that you got less than 40%")
    End If
End Sub
```

4.10 (Variable table)

Variable name	Variable type	used to store
Max_mark	integer	What the test is out of (e.g. 80)
First_name	string	Student's first name (e.g. Albert)
Surname	string	Student's surname (e.g. Einstein)
Mark	integer	Student's actual mark (e.g. 63)
Percent	single	Student's percentage (e.g. 53.7)
Grade	string	Student's grade (e.g. D)
Init	string	Student's first initial (e.g. A)
Display_name	string	Name to be displayed (e.g. A. Einstein)

4.11 (Mark_grader v2 – solution)

```
Select Case Num
   Case Is >= 80
      exam_grader = "A+"
   Case Is >= 70
      exam_grader = "A"
   Case Is >= 60
      exam_grader = "B"
   Case Is >= 50
      exam_grader = "C"
   Case Is >= 40
      exam_grader = "D"
   Case Else
      exam_grader = "fail"
End Select
```

4.12 (Postage problem – solution)

There are many possible solutions. Here are 3 versions of the branching section of the code:

(a) using multiple Ifs with complex conditions:

```
' calculate cost
  If weight < 60 Then cost = 0.28
  If weight >= 60 And weight < 100 Then cost = 0.42
  If weight >= 100 And weight < 150 Then cost = 0.60
  ... etc.
  ... etc.
  If weight >= 1000 Then cost = 3.45 + (0.86 * (INT((weight - 1000)/250)) + 1)
```

(b) using nested Ifs:

```
' calculate cost
  If weight < 60 Then
    cost = 0.28
    ElseIf weight < 100 Then
       cost = 0.42
       ElseIf weight < 150 Then
          cost = 0.60
          ... etc.
          ... etc.
          Else
            cost = 3.45 + (0.86 * (INT((weight - 1000)/250)) + 1)
  End If
```

(c) using Select Case

```
' calculate cost
  Select Case weight
    Case Is < 60
      cost = 0.28
    Case Is < 100
      cost = 0.42
    Case Is < 150
      cost = 0.60
    ... etc
    ... etc
    Case Else
       cost = 3.45 + (0.86 * (INT((weight - 1000)/250)) + 1)
  End Select
```


Answers to Topic 5 exercises

5.5 (Modifications (1))

```
For Counter = 1 to 12
  Console.WriteLine("Goodbye")
Next

For Counter = 1 to 200
  Console.WriteLine("I must work harder")
Next

For Counter = 1 to 100
  Console.WriteLine("This is very easy")
Next
```

5.5 (Modifications (3))

```
Sub Main ( )
  ' improved For... Next example

  Dim counter As Integer
  Dim message1, message2 As String
  Dim how_many As Integer

  message1 = InputBox("Message required (first line)...")
  message2 = InputBox("Message required (second line)...")
  how_many = InputBox("How many repetitions?")

  For counter = 1 To how_many
    Console.WriteLine(message1)
    Console.WriteLine(message2)
    Console.WriteLine( )
  Next
End Sub
```

produces a blank line

5.6 (For...next table)

Coding used	Results
For counter = 1 To 100	1 2 3 4... 99 100
For counter = 1 To 9999	1 2 3 4... 9998 9999
For counter = 1 To 100 Step 2	1 3 5 7... 97 99
For counter = 2 To 100 Step 2	2 4 6 8... 98 100
For counter = 0 To 100 Step 10	0 10 20 30... 90 100
For counter = -10 To 10 Step 5	-10 -5 0 5 10
For counter = 100 To 1 Step -5	100 95 90 85... 10 5
For counter = 0 To 5 Step 0.5	0 0.5 1.0 1.5 2.0... 4.5 5.0

5.6 (Activity)

(a) For counter = 3 To 36 Step 3
(b) For counter = 0 To 99 Step 9
(c) For counter = 10 To 0 Step –1
(d) For counter = 0 To 4.5 Step 0.75
(e) For counter = 50 To -50 Step –10
(f)
```
For counter = 1 To 10
  Console.WriteLine(counter^2)
```
(g)
```
For counter = 1 To 11
  Console.WriteLine(2^counter)
```


5.6 (General purpose counting program – solution)

```
Dim lower, upper, step_size as Single
Dim counter as Single

lower = InputBox("Enter the lower limit: ")
Upper = InputBox("Enter the upper limit: ")
Stepsize = InputBox("Enter the step size: ")

For counter = lower To upper Step step_size
  Console.WiteLine (counter)
Next
```

5.6 (Multiplication tables – solution)

```
Dim multiplier, counter, answer As Integer
Dim message As String

multiplier = InputBox("Enter the multiplier: ")

For counter = 1 To 12
 answer =  counter * multiplier
 message = counter & " x " & multiplier & " = " & answer
 Console.WriteLine (message)
Next
```

5.7 (For... Next solutions)

Times Table (advanced version)

```
For counter = 1 To maximum
 answer =  counter * multiplier
 message = multiplier & " times " & counter & " equals " & answer
 Console.WriteLine (message)
Next
```

Cost and weight calculator

```
For counter = 0 To 2.0 Step 0.1
 answer =  counter * kilo_price
 message = counter & "kg costs £" & answer
 Console.WriteLine (message)
Next
```

Cubic numbers

```
For counter = first_num To last_num
 answer =  counter^3
 message = counter & " cubed = " & answer
 Console.WriteLine (message)
Next
```

Quadratic function calculator

```
For counter = -5 To 5 Step step_size
 answer =  (3 * (counter^2)) + 4
 message = "x = " & counter & "     >>>>>>>>>  y = " & answer
 Console.WriteLine (message)
Next
```


5.9 (Improvements 1 and 2 – solution)

1

```
Sub Main ( )
  ' generates a question to the user
  ' and waits for the correct answer

  Dim user_answer As Integer
  Dim correct_answer As Integer
  Dim counter As Integer

  counter = 0
  correct_answer = 4
```

2

```
  Do
    user_answer = InputBox("What is 2 + 2?")
    counter = counter + 1
    If user_answer <> correct_answer Then MsgBox("Wrong, try again!")
  Loop Until user_answer = correct_answer

  MsgBox ("You took " & counter & " tries to get that right")
End Sub
```

5.10 (Random numbers tasks)

Single dice throw	Double dice throw
Sub Main ( ) ' generates a single dice throw Dim number As Integer Randomize number = Int (Rnd * 6) + 1 Console.WriteLine (number) End Sub	Sub Main ( ) ' generates a double dice throw Dim dice1, dice2 As Integer Dim answer As Integer Randomize dice1 = Int (Rnd * 6) + 1 dice2 = Int (Rnd * 6) + 1 answer = dice1 + dice2 Console.WriteLine (dice1) Console.WriteLine (dice2) Console.WriteLine (answer) End Sub

5.11 (Arithmetic Tester v.3 – solution)

```
Sub Main ( )
 ' generates 5 random question to the user
 ' and waits for the correct answer

 Dim user_answer As Integer
 Dim correct_answer As Integer
 Dim first as Integer
 Dim second as Integer
 Dim counter as Integer
 Dim question as Integer

 Randomize
 For question = 1 To 5
   first = Int(Rnd * 12) + 1
   second = Int(Rnd * 12) + 1
   counter = 0
   correct_answer = first * second

   Do
      user_answer = InputBox("What is " & first & " x " & second & "?", "Question" & question)
      counter = counter + 1
      If user_answer <> correct_answer Then MsgBox("Wrong, try again!")
   Loop Until user_answer = correct_answer
   Select Case counter
    Case Is 1
      MsgBox("Well done – right first time!")
    Case Else
      MsgBox("Well done!  You took " & counter & " tries")
   End Select
 Next
End Sub
```

change 6 to 5 for number of questions

change 10 to 12 for maximum random number

*Change + to * for multiplication (and use x in the prompt)*

you could use If... Then... Else, but this is how it could be done using Select... Case.

