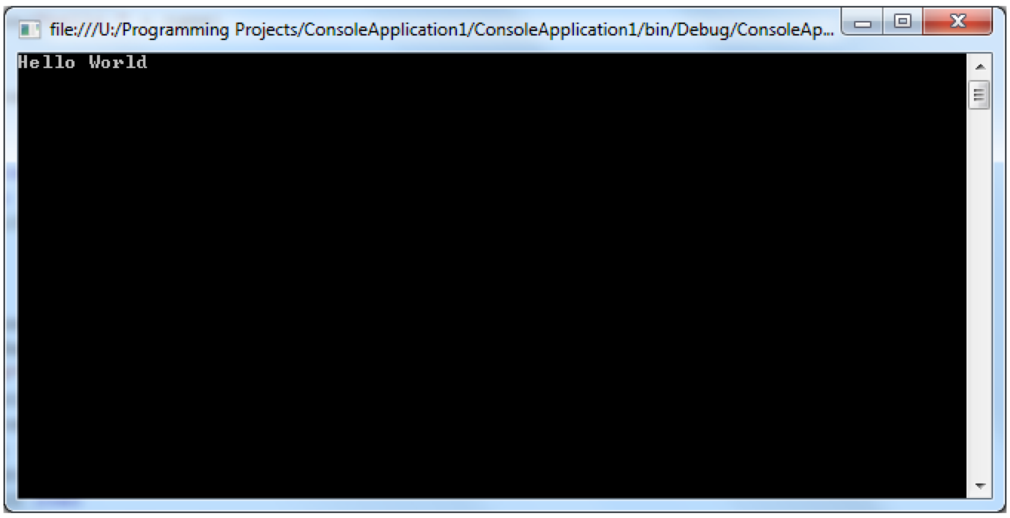
Visual Basic
Console Cook Book

Contents

1. Introduction

These notes describe how to use Microsoft's Visual Basic. While this is a full windows environment, these notes will describe the basic programming concepts and console applications

The console is a basic text window that allows text to be displayed and data to be entered. While it is not true windows programming, it is excellent to get you going with programming.


file:///U:/Programming Projects/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleAp...

```
Hello World
```

Figure 1 - Console

Visual Basic Express

If you don't have either Visual Studio then you can use the free Visual Basic Express Edition from Microsoft. This can be downloaded from the following location:

http://www.microsoft.com/express/Downloads/#2008-Visual-Basic

Computer System

Programming structures

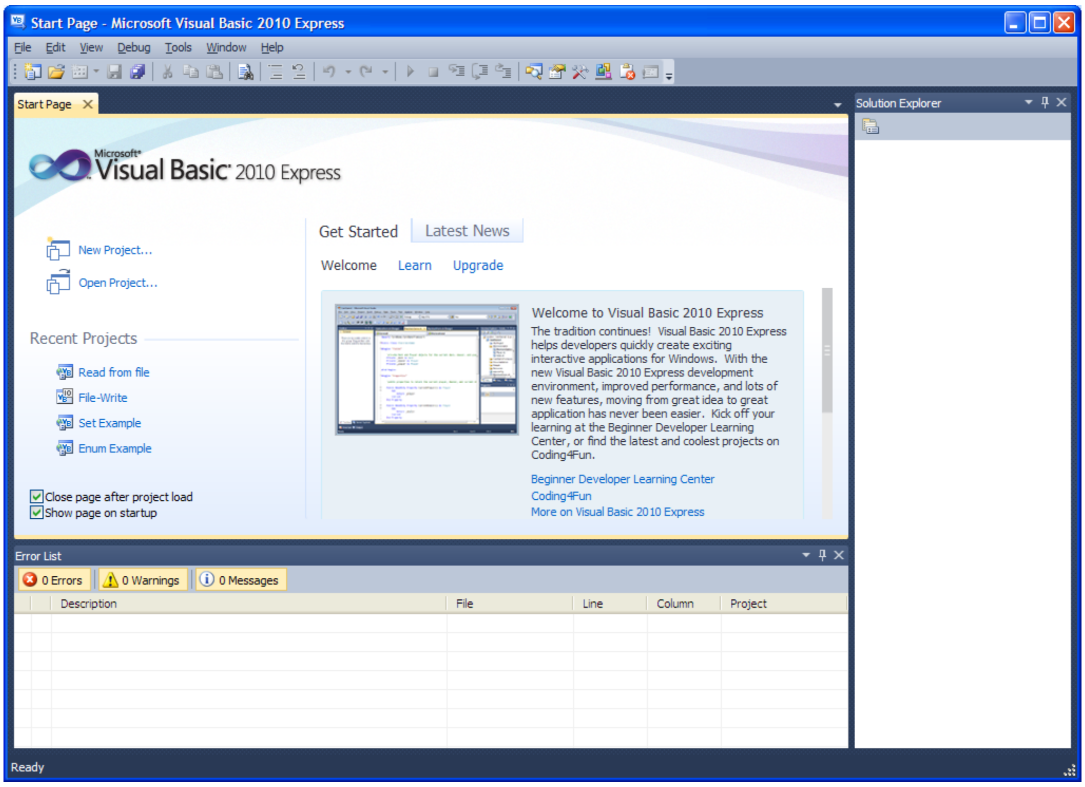
Any program is made from four programming structures:

- Sequence – performing each statement in order from first to last.
- Selection – deciding which statements to execute depending on a condition
- Iteration – repeating statements

Another key aspect of programming is Assignment

Creating a Console Application

Launch your Visual Basic .NET or Visual Studio software. When the software first loads, you'll see a screen something like this one:



Welcome to Visual Basic 2010 Express

The tradition continues! Visual Basic 2010 Express helps developers quickly create exciting interactive applications for Windows. With the new Visual Basic 2010 Express development environment, improved performance, and lots of new features, moving from great idea to great application has never been easier. Kick off your learning at the Beginner Developer Learning Center, or find the latest and coolest projects on Coding4Fun.

Figure 2 - Starting Visual Basic

There's a lot happening on the start page. But basically, this is where you can start a new project, or open an existing one. The first Tab, Start Page, is selected.

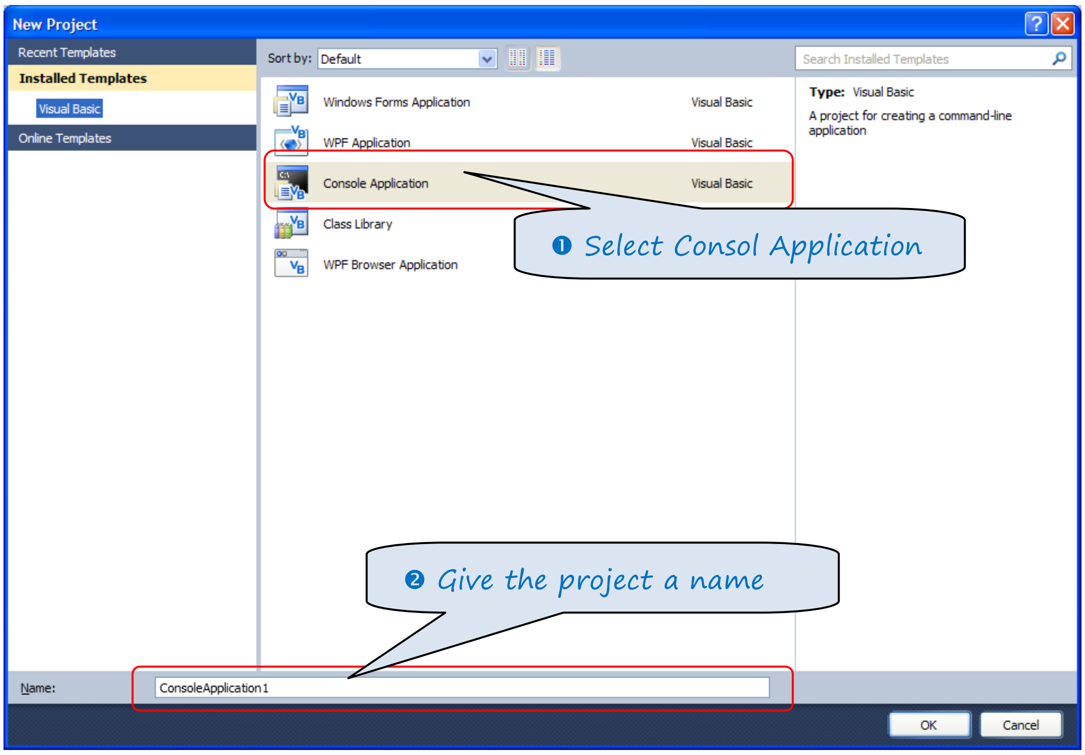
To create a new project, a program in Visual Basic is called a project as it contains many elements

1. Select the file Menu
2. Select New Project

Within Visual Basic, you can create many different types of programs. We will get started by using the console application.

The new Project dialogue box appears, allowing you to select pre-created templates. From this windows,

1. Select the Consol Application template
2. Enter Hello World in the projects name box


New Project
Recent Templates
Installed Templates
Visual Basic
Online Templates
Sort by: Default
Search Installed Templates
Windows Forms Application Visual Basic
WPF Application Visual Basic
Console Application Visual Basic
Class Library
WPF Browser Application
❶ Select Consol Application
❷ Give the project a name
OK
Cancel

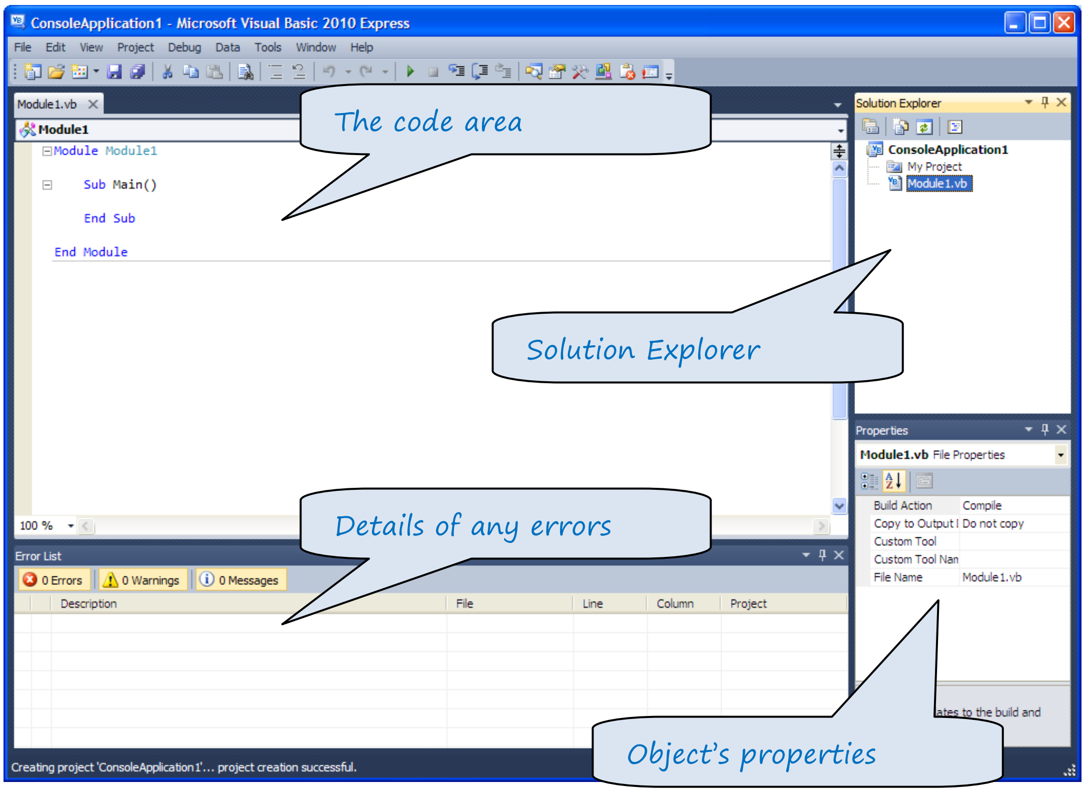
Type: Visual Basic

A project for creating a command-line application

Name: ConsoleApplication1

Figure 3 - Types of Projects

The project environment now shows your program



```
Module Module1

    Sub Main()

    End Sub

End Module
```

Build Action	Compile
Copy to Output I	Do not copy
Custom Tool	
Custom Tool Nan	
File Name	Module1.vb

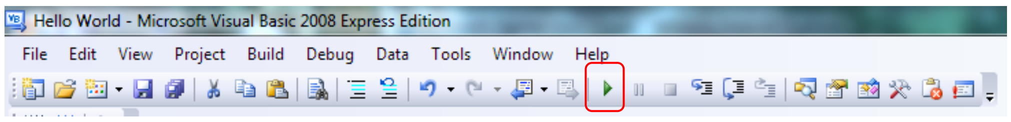
Creating project 'ConsoleApplication1'... project creation successful.

Figure 4 - Development Environment

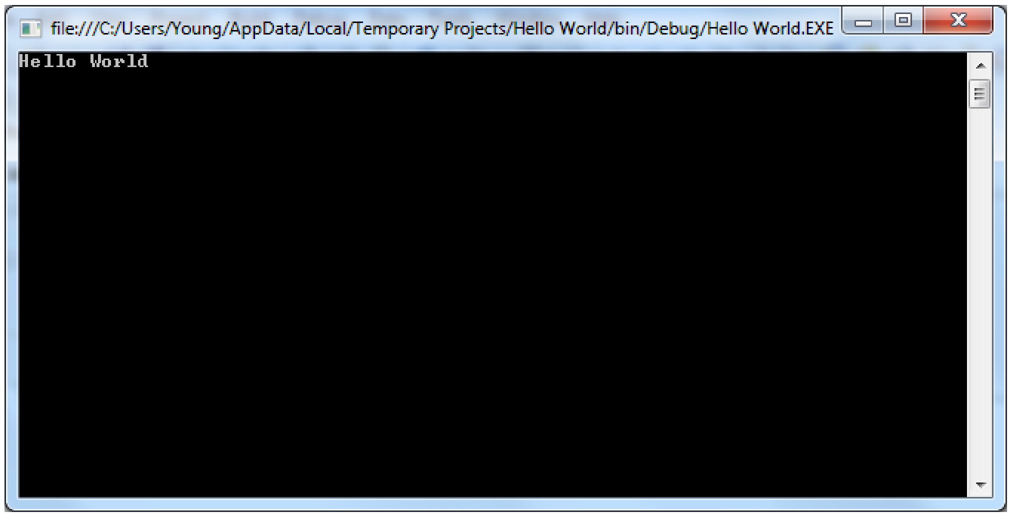
Enter the following code EXACTLY as shown below.

```
Module Module1

    Sub Main()
        Console.WriteLine("Hello World")
        Console.ReadLine()
    End Sub

End Module
```

Press the run button



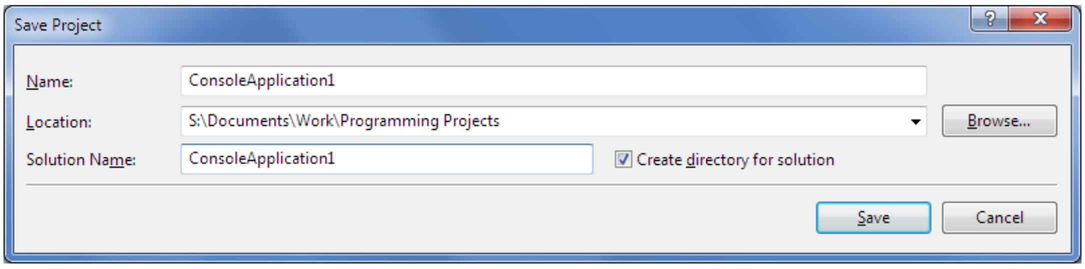
You will see the results in a console window


file:///C:/Users/Young/AppData/Local/Temporary Projects/Hello World/bin/Debug/Hello World.EXE

```
Hello World
```

Saving your Project

There are many ways for saving a projects created in VB.net, bur the best and recommended is to select File →Save All

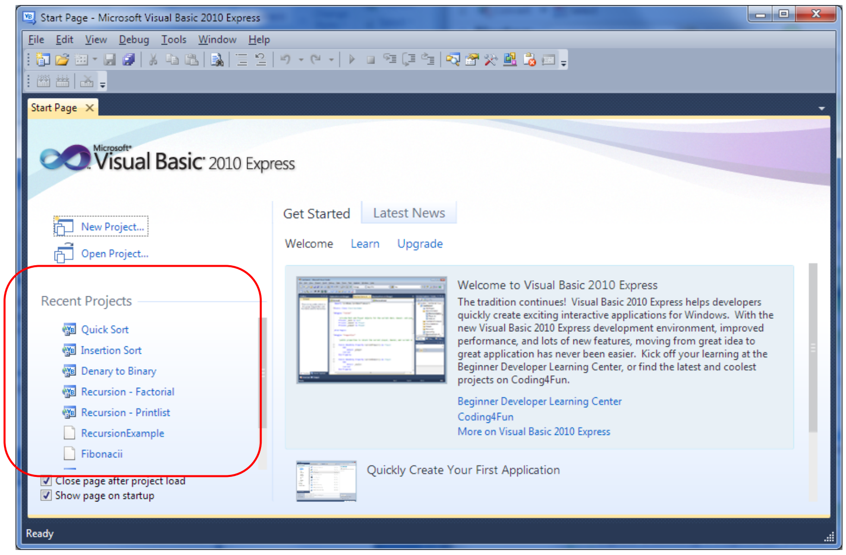


Name: ConsoleApplication1
Location: S:\Documents\Work\Programming Projects
Solution Name: ConsoleApplication1
☑ Create directory for solution

NOTE: if you are saving the projects in school, you MUST have it in the
Visual Studio 2010/projects
folder in your user area. If you save it anywhere else, the program will not run

Opening a Project

In VB.net, you can select a recent project from the start page:


Start Page - Microsoft Visual Basic 2010 Express
File Edit View Debug Tools Window Help
Start Page
Microsoft Visual Basic 2010 Express
New Project...
Open Project...
Get Started
Latest News
Welcome
Learn
Upgrade
Ready

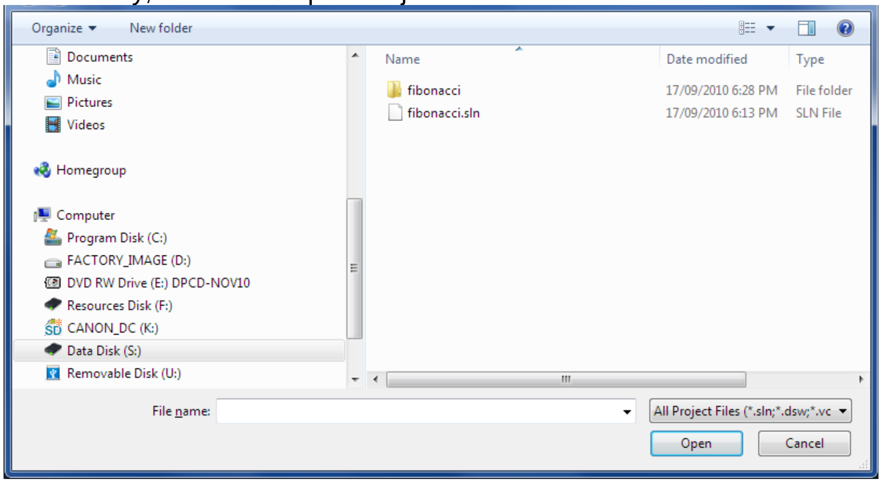
Recent Projects
- Quick Sort
- Insertion Sort
- Denary to Binary
- Recursion - Factorial
- Recursion - Printlist
- RecursionExample
- Fibonacii

☑ Close page after project load
☑ Show page on startup

Welcome to Visual Basic 2010 Express
The tradition continues! Visual Basic 2010 Express helps developers quickly create exciting interactive applications for Windows. With the new Visual Basic 2010 Express development environment, improved performance, and lots of new features, moving from great idea to great application has never been easier. Kick off your learning at the Beginner Developer Learning Center, or find the latest and coolest projects on Coding4Fun.
Beginner Developer Learning Center
Coding4Fun
More on Visual Basic 2010 Express

Quickly Create Your First Application

Alternatively, use File →Open Project



Name	Date modified	Type
fibonacci	17/09/2010 6:28 PM	File folder
fibonacci.sln	17/09/2010 6:13 PM	SLN File

File name:
All Project Files (*.sln;*.dsw;*.vc

2. Variables

What most programs end up doing is storing things in the computer's memory, and manipulating this store.

If you want to add two numbers together, you put the numbers into storage areas and "tell" Visual Basic to add them up. But you can't do this without variables.

So a variable is a storage area of the computer's memory. Think of it like this: a variable is an empty cardboard box. Now, imagine you have a very large room, and in this room you have a whole lot of empty cardboard boxes. Each empty cardboard box is a single variable. To add two numbers together, write the first number on a piece of paper and put the piece of paper into an empty box. Write the second number on a piece of paper and put this second piece of paper in a different cardboard box.

Now, out of all your thousands of empty cardboard boxes two of them contain pieces of paper with numbers on them. To help you remember which of the thousands of boxes hold your numbers, put a sticky label on each of the two boxes. Write "number1" on the first sticky label, and "number2" on the second label.

What have we just done? Well, we've created a large memory area (the room and the cardboard boxes), and we've set up two of the boxes to hold our numbers (two variables). We've also given each of these variables a name (the sticky labels) so that we can remember where they are.

In Visual Basic, variable are created and used as shown below:

```
Dim number1 As Integer
Dim number 2 As Integer

number1 = 3
number2 = 5
```

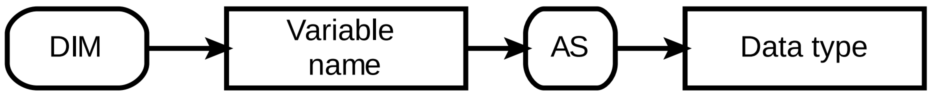
In this code, the 1st and 2nd lines declare two variables; number1 and number2. These variables can only store whole numbers (integers)

The 3rd and 4th lines assign numbers to the variables; number1 stores 3 and number2 stores 5.

Declaring Variables

The process of creating a variable is called declaring a variable. Each declaration needs 4 things:

- **DIM** keyword
- Variable name
- **AS** keyword
- Variable data type



A variable can be called any combination of letters and numbers as long as

- The first character of the variable name is a letter and not a number
- The variable Is one word and doesn't contain spaces.
- Optionally contain _ (underscore)

Which are valid variable name?

Tempname

1st_score

Score

Temp score

It is good practice to declare all your variables, before you use them, at the start of each procedure, module or function.

Declaring Multiple Variables

To save time multiple variables of the same type can be declared with the dame DIM statement.

```
Dim index As Integer
Dim grade As Integer
Dim counter As Integer
```

The three declarations above can be rewritten as one declaration:

```
Dim index, grade, counter As Integer
```

Constants

Just like variables, constants are "dataholders". They can be used to store data that is needed at runtime.

In contrast to variable, the content of a constant can't change at runtime, it has a constant value.

Before the program can be executed (or compiled) the value for a constant must be known.

```
Sub Main()
    'Example of creating constants
    'Written by KPY
    'Date 12/2/2020


    Const pi As Double = 3.1415
    'create a constant called pi with a value 3.1415
    Dim radius As Double = 10
    'creates a constant called radius with a value 10
    Dim circumference As Double = radius * 2 * pi
    'Creates a constant with a calucualtion
    Dim area As Double = radius ^ 2 * pi
    'creating a constant with a calculation
    Console.WriteLine("Circle Circumference : " & circumference)
    Console.WriteLine("Circle Area          : " & area)

    Console.ReadLine()

    End Sub
```

Data types

The following table shows the Visual Basic data types, their supporting common language runtime types, their nominal storage allocation, and their value ranges.

Basic Data Types

A variable can store one type of data. The most used data types are:

Type	Description
Integer	Stores a whole number, e.g. 78
double	Stores a decimal number, e.g. 74.23754
Char	Stores one character, e.g A
string	Stores text, e.g. Hello
Boolean	Stores True or False

Full list of Data Types

Visual Basic type	Nominal storage allocation	Value range
Boolean	Depends on implementing platform	True or False
Byte	1 byte	0 through 255 (unsigned)
Char (single character)	2 bytes	0 through 65535 (unsigned)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	16 bytes	0 through +/- 79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) † with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest nonzero number is +/-0.0000000000000000000000000001 (+/-1E-28) †
Double (double-precision floating-point)	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324 † for negative values; 4.94065645841246544E-324 through 1.79769313486231570E+308 † for positive values
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long (long integer)	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807 (9.2...E+18 †) (signed)
Object	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	1 byte	-128 through 127 (signed)

Short (short integer)	2 bytes	-32,768 through 32,767 (signed)
Single (single-precision floating-point)	4 bytes	-3.4028235E+38 through -1.401298E-45 † for negative values; 1.401298E-45 through 3.4028235E+38 † for positive values
String (variable-length)	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	8 bytes	0 through 18,446,744,073,709,551,615 (1.8...E+19 †) (unsigned)
User-Defined (see page 42)	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members
UShort	2 bytes	0 through 65,535 (unsigned)

† In *scientific notation*, "E" refers to a power of 10. So 3.56E+2 signifies *3.56 x* 10^2 or 356, and 3.56E-2 signifies *3.56 /* 10^2 or 0.0356.

Examples

To create a variable to store text, such as your name:
```
        Dim name As String
```

To create a variable to store a valid date, such as Date of Birth:
```
        Dim DOB As Date
```

To create a variable to store a whole number:
```
        Dim mark As Integer
```

To create a variable to hold any number, including decimal numbers:
```
        Dim temperature As Single
```

To create a variable that stores only True or False
```
        Dim pass As Boolean
```

Which of these variable declarations are correct?

DIM counter AS INTEGER

DIM My Score AS INTEGER

DIM grade AS NUMBER

DIM name AS TEXT

DIM school as string

3. Input & output

Create a new console application and enter the following program:

```
Module Module1

    Sub Main()
        Dim name As String

        Console.WriteLine("Hello, what is your name?")
        name = Console.ReadLine()
        Console.Write("Hello ")
        Console.WriteLine(name)
        Console.ReadLine()
    End Sub

End Module
```

When you have double checked that you have entered the program exactly as above, run it.

This is what the program does:

```
Dim name As String
```

This statement creates a string variable called name

```
Console.WriteLine("Hello, what is your name?")
```

This displays the text within the quotation marks (“) to the console window. In this case it displays *Hello, what is your name.* Notice how it doesn’t display the quotation marks but does add a new line.

```
name = Console.ReadLine()
```

The `readline()` command lets the user type in some text and when the enter key is pressed, will assign it to the variable *name*.

```
Console.Write("Hello ")
```

This displays the text inside the quotation marks but does not give a new line afterwards.

```
Console.WriteLine(name)
```

Notice that there are no quotation marks in this statement? This displays the contents of the variable *name* with a newline at the end

Programming Projects

⌨ Write a console application that asks for your forename and then your surname and then displays both names on the same line.

Writing multiple values

The technical term for joining things together is concatenation. Concatenation operators join multiple strings into a single string. There are two concatenation operators, + and & as summarized below:

Operator	Use
+	String Concatenation
&	String Concatenation

```
Module Module1

    Sub Main()
        Dim name As String
        Dim grade As Integer

        Console.WriteLine("Enter your name")
        name = Console.ReadLine()
        Console.WriteLine("Enter your grade")
        grade = Console.ReadLine()

        Console.WriteLine(name & "'s grade is " & grade)
        Console.ReadLine()
    End Sub

End Module
```

There is a single apostrophe after the quotation mark

Write a console application that asks for your forename and then your surname and then displays both names on the same line.

Write another console application that asks for a name and then a numerical grade (which is a whole number)

4. Adding Comments

Your code should be self documenting this means that you should use

- meaningful and descriptive variable and constant names
- have a description of each module or subroutine
- Who write the program
- Date of the program creation

In Visual Basic, a comment starts the an apostrophe

```
Module Module1
    'Program to concatenate variables
    'written by KPY
    'Date 12/2/2011

    Sub Main()
        'Declaring my variables
        Dim name As String
        Dim grade As Integer

        'Entering the data
        Console.WriteLine("Enter your name")
        name = Console.ReadLine()
        Console.WriteLine("Enter your grade")
        grade = Console.ReadLine()

        'Displaying the results
        Console.WriteLine(name & "'s grade is " & grade)
        Console.ReadLine()
    End Sub

End Module
```

5. Calculations

Calculations are done as part of an assignment:

```
Module Module1
    'Program to calculate the sum and product of two numbers
    'written by KPY
    'Date 12/2/1020

    Sub Main()
        Dim number1 As Integer
        Dim number2 As Integer
        Dim sum As Integer
        Dim product As Integer


        Console.WriteLine("Enter number 1")
        number1 = Console.ReadLine()

        Console.WriteLine("Enter number 2")
        number2 = Console.ReadLine()

        sum = number1 + number2
        product = number1 * number2

        Console.Write("the sum is ")
        Console.WriteLine(sum)

        Console.Write("the product is ")
        Console.WriteLine(product)

        Console.ReadLine()

    End Sub

End Module
```

Notice how the above program take the form

- Input
- Process
- Output

Operators

Operators perform some action on operands (variables)

Operator	Use
^	Exponentiation
-	Negation (used to reverse the sign of the given value, exp -intValue)
*****	Multiplication
/	Division
\	Integer Division
Mod	Modulus Arithmetic
+	Addition
-	Subtraction

Examples

```
Dim z As Double
z = 23 ^ 3
' The preceding statement sets z to 12167 (the cube of 23).
```

```
Dim k As Integer
k = 23 \ 5
' The preceding statement sets k to 4.
```

Integer division is carried out using the \ Operator. Integer division returns the quotient, that is, the integer that represents the number of times the divisor can divide into the dividend without consideration of any remainder.

Both the divisor and the dividend must be integral types (**SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long**, and **ULong**) for this operator. All other types must be converted to an integral type first. The following example demonstrates integer division.

```
Dim x As Integer = 100
Dim y As Integer = 6
Dim z As Integer
z = x Mod y
' The preceding statement sets z to 4.
```

```
Dim a As Double = 100.3
Dim b As Double = 4.13
Dim c As Double
c = a Mod b
' The preceding statement sets c to 1.18.
```

Modulus arithmetic is performed using the `Mod` Operator . This operator returns the remainder after dividing the divisor into the dividend an integral number of times. If both divisor and dividend are integral types, the returned value is integral. If divisor and dividend are floating-point types, the returned value is also floating-point. The following example demonstrates this behaviour.

When doing calculations, you must make sure the variable on the left of the = can store the result of the calculation on the right.

Math Functions

Visual Basic provides support for handling Mathematical calculations. Math functions are stored in System.Math class. The following table gives some of the most useful functions.

Function	Use
Math.Abs()	Returns the absolute value. Math.Abs(-10) returns 10.
Math.Ceiling()	Returns an integer that is greater than or equal to a number. Math.Ceiling(5.333) returns 6.
Fix()	Returns the integer portion of a number. Fix(5.3333) returns 5.
Math.Floor()	Returns an integer that is less than or equal to a number. Fix(5.3333) returns 5.
Int()	Returns the integer portion of a number. Int(5.3333) returns 5.
Math.Max()	Returns the larger of two numbers. Math.Max(5,7) returns 7.
Math.Min()	Returns the smaller of two numbers. Math.Min(5,7) returns 5.
Math.Pow()	Returns a number raised to a power. Math.Pow(12,2) returns 144.
Rnd()	Returns a random number between 0 and 1. Used in conjunction with Randomizestatement to initialize the random number generator.
Math.Round()	Rounds a number to a specified number of decimal places. Rounds up on .5. Math.Round(1.1234567,5) returns 1.12346.
Math.Sign()	Returns the sign of a number. Returns -1 if negative and 1 if positive. Math.Sign(-5) returns -1.
Math.Sqrt()	Returns the square root of a positive number. Math.Sqrt(144) returns 12.

Math.Sin()
Math.Cos()
Math.Tan()
Math.Asin()
Math.Acos()
Math.ATan()
Math.Sinh()
Math.Cosh()
Math.Tanh()
	Trigonometry function.
	Math.sin(45) returns 0.0850903524534118
	
	Note: the result is in radians. To convert degrees to radians you multiply by 2 *pi / 360

Math.Log()	Returns natural (base e) logarithm
Math.log10	Returns the base 10 logarithm

Formatting numbers

If single or double numbers are displayed, you may want to have control over how that number is formatted, e.g. show 2 decimal places.

The Format function converts a value to a text string and gives you control over the string's appearance. For example, you can specify the number of decimal places for a numeric value, leading or trailing zeros, currency formats, and portions of the date.

For a full list of the formatting symbols, see page 89.

Formatting Examples

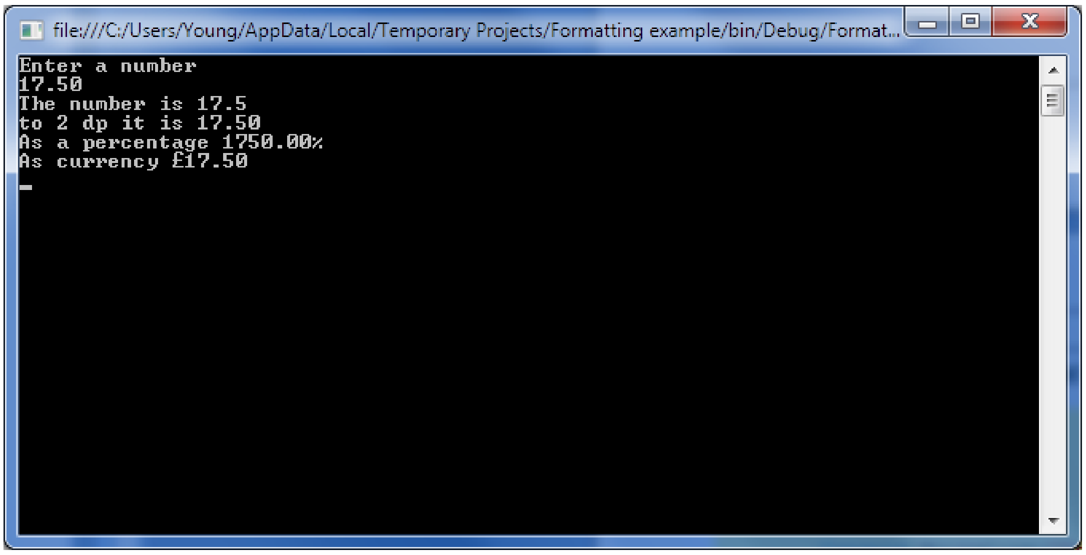
Format syntax	**Result**
Format(8315.4, "00000.00")	08315.40
Format(8315.4, "#####.##")	8315.4
Format(8315.4, "##,##0.00")	8,315.40
Format(315.4, "$##0.00")	$315.40
Format(7, "0.00%")	700.00%

Format syntax	**Result**
Format("This Is A Test", "<")	this is a test
Format("This Is A Test", ">")	THIS IS A TEST

Format syntax	**Result**
Format(Now, "m/d/yy")	1/27/10
Format(Now, "dddd, mmmm dd, yyyy")	Wednesday, January 27, 2010
Format(Now, "d-mmm")	27-Jan
Format(Now, "mmmm-yy")	January-10
Format(Now, "hh:mm AM/PM")	07:18 AM
Format(Now, "h:mm:ss a/p")	7:18:00 a
Format(Now, "d-mmmm h:mm"	27-January 7:18
Format(Now, "d-mmmm-yy")	27-January-10
Format(Now, "d mmmm")	27 January
Format(Now, "mmmm yy")	January 10
Format(Now, "hh:mm AM/PM")	08:50 PM
Format(Now, "h:mm:ss a/p")	8:50:35 p
Format(Now, "h:mm")	20:50
Format(Now, "h:mm:ss")	20:50:35
Format(Now, "m/d/yy h:mm")	1/27/93 20:50

Sample Program using format

```
Module Module1

    Sub Main()
        Dim num1 As Single

        Console.WriteLine("Enter a number")
        num1 = Console.ReadLine()
        Console.WriteLine("The number is " & num1)
        Console.WriteLine("to 2 dp it is " & Format(num1, "####.00"))
        Console.WriteLine("As a percentage " & Format(num1, "####.00%"))
        Console.WriteLine("As currency " & Format(num1, "£####.00"))
        Console.ReadLine()

    End Sub

End Module
```


file:///C:/Users/Young/AppData/Local/Temporary Projects/Formatting example/bin/Debug/Format...

```
Enter a number
17.50
The number is 17.5
to 2 dp it is 17.50
As a percentage 1750.00%
As currency £17.50
_
```

Programming Projects

Write a program that will read in three integers and display the sum.

Write a program that will read two integers and display the product.

Enter the length, width and depth of a rectangular swimming pool. Calculate the volume of water required to fill the pool and display this volume.

Write a program that will display random numbers between 1 and 6 until a six is generated.

Write a program that will display six random numbers between 5 and 10.

Challenging Projects

x \ y calculates how many times y divides into x, for example 7 DIV 3 is 2. x MOD y calculates the remainder that results after division, for example 7 MOD 3 is 1. Write a program that will read in two integers Number1 and Number2. Using \ and MOD, your program should display the whole number part and the remainder of dividing Number1 by Number2. Make the display easy to understand for the user.

Write a program to enter an amount of money as a whole number, for example £78, and display the minimum number of £20, £10, £5 notes and £2 and £1 coins that make up this amount.

For example, the value £78 would give 3 twenty pound notes, 1 ten pound note, 1 five pound note, 1 two pound coin and 1 one pound coin.

Programming Projects

Write a program that will ask the user for their first name. The program should then concatenate the name with a message, such as 'Hello Fred. How are you?' and output this string to the user.

Write a program that asks the user to enter two double numbers and displays the product of these two numbers to 2 decimal places, with user-friendly messages. (tip: see page 19 for examples)

Calculate the area of a rectangle where the length and breadth are entered by the user.

Calculate the area and circumference of a circle, where the radius is entered into an edit box. The radius and area may not always be integer values.

Create an automatic percent calculator that a teacher could use after an exam. The teacher will enter the total number of marks and the raw score. The form them calculates the percentage (to two decimal places).

Create a currency converter where the number of pounds is entered and the procedure calculates the number of Euros. Assume the exchange rate of £1 = €1.15.
Adjust your program to allow any exchange rate.

Try to adapt program 2 to deal with a cylinder.

Area of an open cylinder is 2πrh.
Area of a closed cylinder is 2πr(h + r)
Volume of a cylinder is πr2h

Convert a temperature entered in Celsius to Fahrenheit where

$$F = \left(\frac{9}{5} \times C\right) + 32$$

Write a program to convert a person's height in inches into centimetres and their weight in stones into kilograms. [1 inch = 2.54 cm and 1 stone = 6.364 kg]

Challenging Projects

Write a program to enter the length and width of a rectangular-shaped garden. Calculate the area of the garden and the cost of turfing a lawn if a 1 m border is around the perimeter of the garden. Assume the cost of turf is £10 per square metre. Display the result of these calculations.

Write a program to enter the length, width and depths at the deepest and shallowest ends of a rectangular swimming pool. Calculate the volume of water required to fill the pool, and display this volume.

6. Selection

With the selection structure, some statements are only executed if a condition has been met.

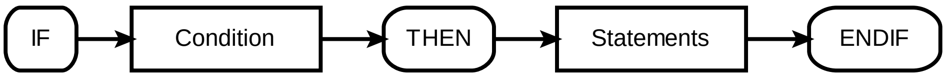
The question asked must be a Boolean question. That is a question that can be answered TRUE or FALSE

Comparison Operators

A comparison operator compares operands and returns a logical value based on whether the comparison is true or not. The table below summarizes them:

Operator	Use
=	Equality
<>	Inequality
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to

The IF THEN statement



```
Module Module1
    'Program to demonstrate the IF THEN statement
    'written by KPY
    'Date 12/2/1020

    Sub Main()
        Dim grade As Integer

        Console.WriteLine("Enter your grade")
        grade = Console.ReadLine()

        If grade > 50 Then
            Console.WriteLine("You have passed")
        End If

        Console.ReadLine()
    End Sub


End Module
```

Complex conditions

A complex condition is when there are a number of questions in the condition, for example:

If the grade is greater than 50 and less than 70 then …..

This would be programmed

```
        If (grade > 50) AND (grade < 70) Then
            Console.WriteLine("You have passed")
        End If
```

When AND is used, all parts of the condition must be true to run the code.

```
        If (grade > 50) OR (grade < 70) Then
            Console.WriteLine("You have passed")
        End If
```

If the logical operator is OR, then one or the other (but not all) need to be true to run the code

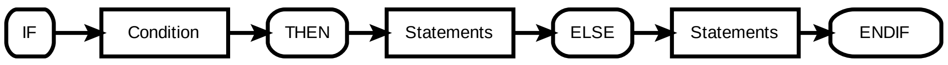
```
        If Not (grade < 75) Then
            Console.WriteLine("You have passed")
        End If
```

Logical Operators

The logical operators compare Boolean expressions and return a Boolean result. In short, logical operators are expressions which return a true or false result over a conditional expression. The table below summarizes them:

Operator	Use
Not	Negation
And	All true (Conjunction)
Or	At least one is true (Disjunction)
Xor	One or the other is true, but not both (Disjunction)

The IF THEN ELSE statement



```
Module Module1
    'Program to demonstrate the IF THEN ELSE statement
    'written by KPY
    'Date 12/2/2010

    Sub Main()
        Dim grade As Integer

        Console.WriteLine("Enter your grade")
        grade = Console.ReadLine()

        If grade > 50 Then
            Console.WriteLine("You have passed")
        Else
            Console.WriteLine("You have failed")
        End If

        Console.ReadLine()
    End Sub

End Module
```

Nested If

Nested if statement helps to check multiple number of conditions. A nested statement is one where statements are inserted inside a statement of the same type.

A nested IF statement has an IF statement inside and IF statement, inside an IF statement …..

This is totally acceptable for a few nested statements, but if there are too may, it becomes very hard to debug and follow the logic of the program.

```
    Sub Main()
        Dim grade As Integer

        Console.WriteLine("Enter a grade")
        grade = Console.ReadLine

        If grade > 80 Then
            Console.WriteLine("Grade A")
        Else
            If grade > 60 Then
                Console.WriteLine("Grade B")

            Else
                If grade > 50 Then
                    Console.WriteLine("Grade C")
                Else
                    Console.WriteLine("Grade U")
                End If
            End If
        End If

        Console.ReadLine()
    End Sub
```

As you can see, after a several nested ifs, the program looks confusing. A better solution would be to re-write the statements using the IF ELSEIF, shown on page 27.

IF ELSEIF statement

Sometimes you may need to test multiple "mutually exclusive" conditions -- a series of conditions only one of which will be true. The If...ElseIf construct tests for as many such conditions as need to be tested. Its general format is shown below.

Notice that each ELSEIF has a condition (unlike the nested IF on the previous page)

```
    Sub Main()
        Dim grade As Integer

        Console.WriteLine("Enter a grade")
        grade = Console.ReadLine

        If grade > 80 Then
            Console.WriteLine("Grade A")
        ElseIf grade > 60 Then
            Console.WriteLine("Grade B")
        ElseIf grade > 50 Then
            Console.WriteLine("Grade C")
        Else
            Console.WriteLine("Grade U")
        End If

        Console.ReadLine()
    End Sub
```

Select Case

The IF statement is useful, but can get clumsy if you want to consider “multi-way selections”. Similarly to the IF statement the Select Case tests the contents of a variable and executes statements depending on its value.

```
Module Module1
    'Program to demonstrate the SELECT CASE statement
    'written by KPY
    'Date 12/2/2010

    Sub Main()
        Dim grade As Integer

        Console.WriteLine("Enter your grade")
        grade = Console.ReadLine()

        Select Case grade
            Case Is > 70
                Console.WriteLine("Grade A")
            Case 55 To 69
                Console.WriteLine("Grade B")
            Case 45 To 54
                Console.WriteLine("grade C")
            Case 41, 43
                Console.WriteLine("grade E+")
            Case 40,42,44
                Console.WriteLine("grade E-")
            Case Else
                Console.WriteLine("grade D")
        End Select

        Console.ReadLine()
    End Sub
End Module
```

The case statement is useful for checking that a value belongs to a list of results

```
    Sub Main()
        Dim month As Integer

        Console.WriteLine("Enter a month")
        month = Console.ReadLine

        Select Case month
            Case 1, 3, 5, 7, 8, 10
                Console.WriteLine("31 days")
            Case 4, 6, 9, 11
                Console.WriteLine("30 days")
            Case 2
                Console.WriteLine("28 days")
            Case Else
                Console.WriteLine("No such month")
        End Select
        Console.ReadLine()
    End Sub
```

It is also useful for checking ranges of results

```
    Sub Main()
        Dim letter As char

        Console.WriteLine("Enter a character")
        letter = Console.ReadLine

        Select Case letter
            Case "A" To "Z"
                Console.WriteLine("Capital letter")
            Case "a" To "z"
                Console.WriteLine("Lower case letter")
            Case "0" To "9"
                Console.WriteLine("Number")
            Case "+", "-", "*", "/"
                Console.WriteLine("Operator")
            Case Else
                Console.WriteLine("symbol")
        End Select
        Console.ReadLine()
    End Sub
```

Programming Projects

Type in the above program into a new console application. Then test your program with the following test data. For each set of test data:

- state the expected results,
- run the program with the test data and
- get a screen shot to show evidence.

Test Data with expected results

- Grade = 10
- Expected result = Grade U

- Grade = 80
- Expected result = Grade A

- Grade = 60
- Expected result = ………………….

- Grade = 50
- Expected result = …………………

- Grade = 110
- Expected result = …………………

- Grade = 70
- Expected result = …………………

- Grade = 45
- Expected result = …………………

Amend the grade program to that an error message is displayed if the grade is greater than 100 or less than 0.

Write a program that will ask for a person's age and then display a message whether they are old enough to drive or not

Write a program that asks the user to enter two numbers and then displays the largest of the two numbers

Write a program that checks whether a number input is within the range 21 to 29 inclusive and then displays an appropriate massage.

Extend the program so a message out of range will cause a message saying whether it is above or below the range

Write a program that asks for three numbers and then displays the largest of the three numbers entered

Reynolds number is calculated using formula (D*v*rho)/mu Where D = Diameter, V= velocity, rho = density and mu = viscosity

Write a program that will accept all values in appropriate units (Don't worry about unit conversion)

- If Reynolds number is < 2100, display Laminar flow,
- If it’s between 2100 and 4000 display 'Transient flow' and
- if more than '4000', display 'Turbulent Flow' (If, else, then...)

Write a program that asks the user for a month number and displays the number of days that month has (ignore leap years for now).

Extend your program to include leap years. A year is a leap year if the year divides exactly by 4, but a century is not a leap year unless it is divisible by 400. For example the year 1996 was a leap year, the year 1900 was not, the year 2000 was a leap year.
(**HINT**: a number is divisible by 4 if the modulus is 0)

Write a program that lets the user enter a number between 1 and 12 and displays the month name for that month number. The input 3 would therefore display March.

Write a program that reads in the temperature of water in a container (in Centigrade) and displays a message stating whether the water is frozen, boiling or neither.

Write a program that asks the user for the number of hours worked this week and their hourly rate of pay. The program is to calculate the gross pay. If the number of hours worked is greater than 40, the extra hours are paid at 1.5 times the rate. The program should display an error message if the number of hours worked is not in the range 0 to 60.

Write a program that accepts a date as three separate integers such as 12 5 03. The program should display the date in the form 12th May 2003.

Adapt your program to interpret a date such as 12 5 95 as 12th May 1995. Your program should interpret the year to be in the range 1931 to 2030.

Challenge Project

Write a program that accepts a date as three separate integers such as 12 5 03 and then calculate is this is a valid date or not.

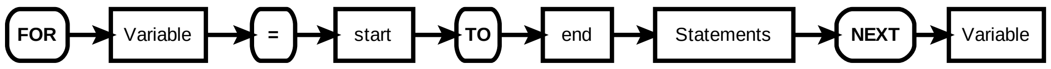
7. Iteration

Iteration is a technical work for doing something over and over again. This saves time by looping round code to do a task multiple times.

A loop is something that goes round and round and round. If I said move your finger around in a loop, you'd know what to do immediately. In programming, loops go round and round and round, too. In fact, they go round and round until you tell them to stop. You can programme without using loops. But it's an awful lot easier with them.

FOR loop

The fore loop repeats statements a set number of time. It uses a variable to count how many time it goes round the loop and stops when it reaches its limit.



```
Module Module1
    'Program to demonstrate the FOR loop
    'written by KPY
    'Date 12/2/1020

    Sub Main()
        Dim index As Integer

        For index = 1 To 20
            Console.WriteLine(index & " times 5 is " & index * 5)
        Next
        Console.ReadLine()
    End Sub

End Module
```

FOR STEP loop

You may not wish to count consecutively through this loop, for example you may wish to count every other number.

```
        Dim index As Integer
        '
        For index = 2 To 10 Step 2
            Console.WriteLine(index)
        Next
        '
        Console.ReadLine()
    End Sub
```

Or you may wish to count backwards

```
    Sub Main()
        Dim index As Integer
        '
        For index = 10 To 0 Step -2
            Console.WriteLine(index)
        Next
        '
        Console.ReadLine()
    End Sub
```

Programming Projects

Write a program that displays the word 'Hello' on the screen 4 times on the same line using the for loop.

Write a program that prompts the user to enter a short message and the number of times it is to be displayed and then displays the message the required number of times.

Write a console application to calculate the 8 times table

Write a console application that will ask for 10 numbers. The program will then display the sum of these numbers and the average.

Write a program that asks for a number, and displays the squares of all the integers between 1 and this number inclusive.

Write a program to display the squares of all the integers from 1 to 12 in two columns headed 'Number' and 'Square of Number'.

Write a program that displays all the numbers between 1 and 10,000. How long did it take to execute?

Write a program that displays all even numbers between 1 and 10,000. How long did it take to execute?

Write a console application that will ask for 10 numbers. The program will then display the max and min value entered (you will need to use an IF statement and variables to hold the max and min values)

n factorial, usually written n!, is defined to be the product of all the integers in the range 1 to n:

n! = 1 * 2 * 3 * 4 * …….*n

Write a program that calculates n! for a given positive n.

Challenging Projects

Write a program that asks for a number and converts this into a binary number. You will need to use \ and mod.

Write a program that shows the first 10 numbers in the Fibonacci series.

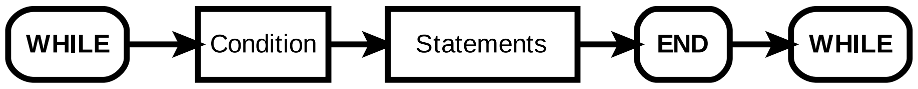
A prime number is a number that can only be divided by the number itself and 1. Write a program that displays all the prime numbers between 2 and 1000.

WHILE loop

The wile look is known as a **test before loop**. The condition is tested before entering the loop, but tested each time it goes round the loop.

The number of times the statements within the loop are executed varies. The test before loop goes round 0 or more times.

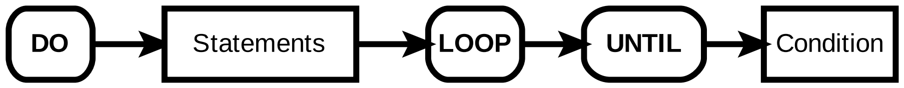
This method is useful when processing files and using “read ahead” data



```
    Sub Main()
        Dim name As String

        name = Console.ReadLine()
        'Test before loop –
        'only enter the loop is name not equal "X"
        While name <> "X"
            Console.WriteLine(name)
            name = Console.ReadLine()
        End While

    End Sub
```

REPEAT loop

The repeat loop is similar to the while loop, but it tests the condition after the statements have been executed once. This means that this test after loop goes round 1 or more times.



```
    Sub Main()
        Dim name As String

        Do
            name = Console.ReadLine()
            Console.WriteLine(name)
        Loop Until name = "X"
    'Test after loop

    End Sub
```

Programming Projects

Write a program that reads in a series of numbers and adds them up until the user enters zero. (This stopping value is often called a **rogue value.)** You may assume that at least 1 number is entered.

Expand your program to display the average as well as the sum of the numbers entered. Make sure you do not count the rogue value as an entry.

Write a program that asks the user for a number between 10 and 20 inclusive and will validate the input. It should repeatedly ask the user for this number until the input is within the valid range.

Make changes to your program so it will also work if the user does not want to type in any numbers and uses the rogue value straight away.

Write a program that displays a conversion table for pounds to kilograms, ranging from 1 pound to 20 pounds [1 kg = 2.2 pounds].

Write a program that asks the user to enter 8 integers and displays the largest integer.

Adapt your program so that the user can type in any number of positive integers. Input will terminate with the rogue value of—l.

Adapt your program so that it will also display the smallest integer.

Adapt your program from if necessary, so that it works if the user types in the rogue value as the first value

Challenge Project

Write a game in which the user guesses what random number between 1 and 1000 the computer has 'thought of, until he or she has found the correct number. The computer should tell the user whether each guess was too high, too low or spot on. (TIP: use the Maths library, which has a random function. See page 18)

8. Array Data Type

An array is a special variable that has one name, but can store multiple values. Each value is stored in an element pointed to by an index.

The first element in the array has index value 0, the second has index 1, etc

One Dimensional Arrays

A one dimensional array can be thought as a list. An array with 10 elements, called names, can store 10 names and could be visualised as this:

index	Element
0	Fred
1	James
2	Tom
3	Robert
4	Jonah
5	Chris
6	Jon
7	Matthew
8	Mikey
9	Jack

This array would be created by

```
    Dim names(9) As String
```

Elements indexed from 0 to 9

The statement:

```
        Console.WriteLine(names(1))
```

Will display James

```
        Console.WriteLine(names(7))
```

Will display Matthew

```
Sub Main()
    'Example of a 1 Dimensional array of 10 elements
    'Written by KPY
    'Date 12/2/2020

    Dim index As Integer
    Dim names(9) As String 'declaring a 10 element array of string
    Dim grades(9) As Integer ' decalaring a 10 element array of
integer

    'Entering 10 names and grades
    For index = 0 To 9
        Console.WriteLine("Enter name " & index)
        names(index) = Console.ReadLine()
        Console.WriteLine("Enter grade for " & names(index))
            grades(index) = Console.ReadLine()
    Next index

    'Displaying the 10 names and grades
    For index = 0 To 9
        Console.WriteLine(names(index) & " has grade " &
grades(index))
    Next index
End Sub
```

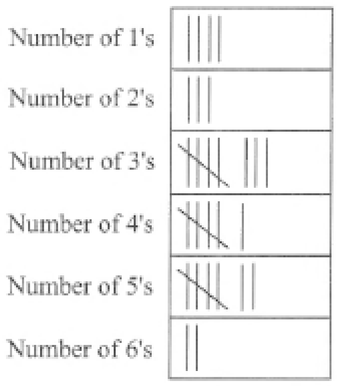
Programming Projects

Write a program that reads 6 names into an array. The program must display the names in the same order that they were entered and then in reverse order.

Make a program that fills an array with 5 elements with values entered by the user.
Print out all elements in order.

Make a program that first asks the user how many values to enter. The program will then fill an array with all entered values. When all values are entered the program prints out the highest entered value and its position (and index) in the array.

We want to simulate throwing a die 30 times and record the scores. If we did this ‘manually' we would end up with a tally chart:



If we use a computer to keep a count of how many times each number was thrown, we could use an integer array (index range 1..6) instead of the tally chart. In general, a die throw will give a score i, and we want to increment the count in the i th element.

```
TallyChart(i) ← TallyChart(i) + 1
```

Write a program to simulate the throwing of a die 30 times. The results of the

We wish to select six random numbers between 1 and 49 with the condition that all the numbers are different. One possible strategy, or algorithm, is.

- Initialise an array by using a for loop to store the values 1 to 49
- Repeatedly select a random element from array until a non-zero value is selected
- Display this value
- Set that element to zero
- Repeat the above three steps until six numbers have been selected.

Write a program to select six unique random numbers between 1 and 49.

Declare two arrays, Student and DoB, to store the name of Students and their dates of birth. For example if Fred is born on 22/12/84, then we could store 'Fred' in Student(1) and '22/12/84' in DoB(1). To find a particular student we can use a repeat loop:

```
Ptr ← 0
  repeat
  Ptr ← Ptr + 1
until (Student[Ptr] = WantedStudent) OR (Ptr = 5)
```

Write a program that stores 5 students' names and dates of birth and then searches for a particular student and displays that student's date of birth and current age. Display a suitable message if the student's details cannot be found.

Write a program which asks the user for the subjects done in each period for each day and then prints out the timetable with suitable headings.

Using a two-dimensional array, write a program that stores the names of ten countries in column 1 and their capitals in column 2. The program should then pick a random country and ask the user for the capital. Display an appropriate message to the user to show whether they are right or wrong.

Expand the program above to ask the user 5 questions and give a score of how many they got right out of 5.

Challenge Projects

Store in a 1-D array a set of 5 place names, and in a 2-D array the distances between the places. Ensure that the order of the places is the same in both arrays. When the names of 2 places are input, the distance between them is displayed. If they are not both in the table, a suitable message should be displayed.

A Latin Square of order n is an n x n array which contains the numbers 1, 2, 3, ..., n such that each row and column contain each number exactly once. For example the following diagram shows a Latin Square of order 4. You can see that each row can be obtained from the previous one by shifting the elements one place to the left.

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

Design and write a program to store such a Latin Square of a size given by the user. The program should also display the Latin Square.

Multi-Dimensional Arrays

A multi-dimensional array can be thought of as a table, each element has a row and column index.

Following example declares a two-dimensional array called `matrix` and would be declared by

```
    Dim matrix(2,3) As Integer
```

Usually we refer to the first dimension as being the rows, and the second dimension as being the columns.

index	0	1	2	3
0	A	B	C	D
1	E	F	G	H
2	I	J	K	L

The following statements would generate the following

```
    Console.WriteLine(matrix(0, 0))
```
Would display A

```
    Console.WriteLine(matrix(2, 1))
```
Would display `J`

```
    Console.WriteLine("first row, first column   : " & matrix(2, 3))
```
Would display `first row, first column   :  L`

```
Sub Main()
    Dim matrix(2,3) As Integer

    matrix(0, 0) = 10
    matrix(1, 0) = 20
    matrix(1, 2) = 30

    Console.WriteLine("first row, first column   : " & matrix(0, 0))
    Console.WriteLine("second row, first column  : " & matrix(1, 0))
    Console.WriteLine("second row, second column : " & matrix(1, 1))
    Console.WriteLine("third row, third column   : " & matrix(1, 2))

    Console.ReadLine()
End Sub
```

Create a program that fills a two-dimensional array with ascending values.
The number of rows, the number of columns and the starting value is determined by the user.

Print out all elements of this array.

9. User defined Data Type

A useful feature of most high level programming languages is the ability to create your own data type.

Think of these as a record structure, where you define each field within the record.

If you want to store information about books, you could create a user defined data type to store all the information about a book:

- ISBN
- Title
- Price
- Year of Publication

In VB.NET you create a user defined data type using the STRUCTURE command.

The section of code below shows how to create a data type called TBook and then create an array of books.

```
Module Module1
    Structure TBook
        Dim ISBN As String
        Dim Title As String
        Dim Price As Decimal
        Dim YearOfPublication As Integer
    End Structure
    Dim books(10) As TBook
:
:
:
```

When creating a user defined type, it is convention to start the data types name with T
TBook is a custom type

Each field is accessed by the statement

```
books(index).ISBN = Console.ReadLine()
```

This subroutine loops entering book details into the array books.

```
  Sub EnterBooks()
        Dim answer As Char
        Dim index As Integer
        index = 0

        Do
            Console.WriteLine("Enter the book's details")
            Console.Write("ISBN :")
            books(index).ISBN = Console.ReadLine()
            Console.Write("Title :")
            books(index).title = Console.ReadLine()
            Console.Write("Price :")
            books(index).price = Console.ReadLine()
            Console.Write("Year of Publication :")
            books(index).YearOfPublication = Console.ReadLine()

            Console.WriteLine("Add another book? Y/N")
            answer = Console.ReadLine().ToUpper
            index = index + 1

        Loop Until (answer = "N" Or index > 10)
        BookCount = index - 1
    End Sub
```

10. Enumeration & Sets

Enumeration

```
Module Module1
    Dim x, y As Integer
    Enum Days
        sun
        Mon
        Tue
        Wed
        Thu
        Fri
        Sat
    End Enum
    Sub Main()
        x = Days.Wed
        y = Days.sun

        Console.WriteLine(x)
        Console.WriteLine(y)
        Console.ReadLine()
    End Sub

End Module
```

Sets

A set is a group of common values. It is the principle of any data processing; a database is a set of records.

They are similar to a two dimensional array. After a set has been created, items can be:

- Added
- Removed
- Sorted

Just as an array, each item can be accessed by its index (index 0 is the first item in the set)

```
Module Module1

    Sub Main()
        Dim set1, set2, set3 As New ArrayList
        Dim initialset() = {3, 4, 5, 6, 10, 11, 12, 14, 20}
        set1.AddRange(initialset)
        Dim initialset2() = {1, 2, 3}
        set2.AddRange(initialset2)
        set1.Add(2)
        set1.Add(7)
        set1.Remove(3)

        Console.WriteLine("unsorted list")
        printlist(set1)

        set1.Sort()
        Console.WriteLine("Sorted list")
        printlist(set1)

        Console.WriteLine("List count is    {0}", set1.Count)
        Console.WriteLine("List capacity of {0}", set1.Capacity)


        If set1.Contains(7) Then
            Console.WriteLine("7 is in the set")
        End If

        If Not set1.Contains(3) Then
            Console.WriteLine("3 is NOT in the set")
        End If

        Console.ReadLine()

    End Sub

    Sub printlist(ByVal mylist As ArrayList)
        For counter = 0 To (mylist.Count - 1)
            Console.WriteLine("Item " & counter & " is " &
mylist(counter))
        Next
    End Sub

End Module
```

11. String Manipulation

If a variable is declared as a string, it can be manipulated in a number of ways.

Using LEN()

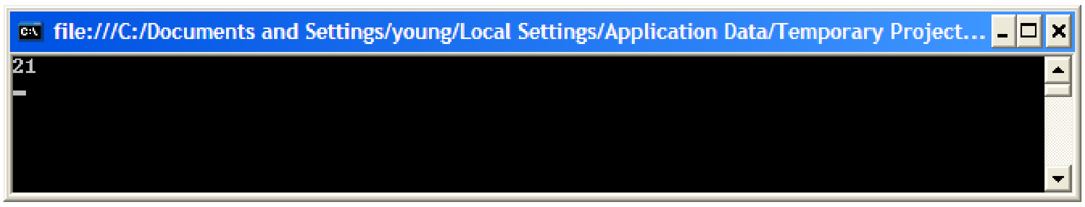
The Length statement returns the number of characters (including spaces and symbols) stored in a string variable.

```
Len(string)
```

Parameter	Description
string	A string expression

```
Module Module1

    Sub Main()
        Dim textstring As String
        Dim stringlength As Integer

        textstring = "This is a text string"
        stringlength = Len(textstring)

        Console.WriteLine(stringlength)
        Console.ReadLine()
    End Sub

End Module
```

Would produce


file:///C:/Documents and Settings/young/Local Settings/Application Data/Temporary Project...

```
21
_
```

Using Left()

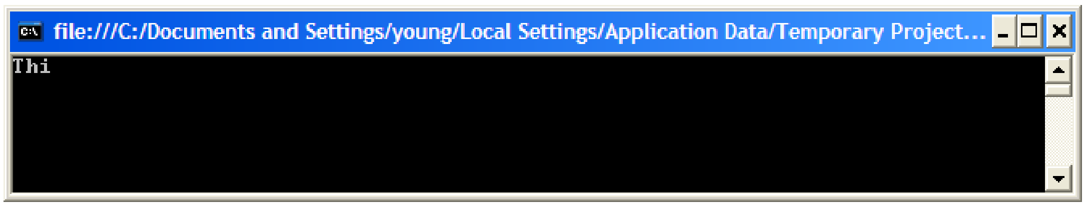
The Left function returns a specified number of characters from the left side of a string or Use the Len function to find the number of characters in a string.

```
Left(string,length)
```

Parameter	Description
string	Required. The string to return characters from
length	Required. Specifies how many characters to return. If set to 0, an empty string ("") is returned. If set to greater than or equal to the length of the string, the entire string is returned

```
Module Module1

    Sub Main()
        Dim textstring As String
        Dim leftstring As String

        textstring = "This is a text string"
        leftstring = Left(textstring, 3)

        Console.WriteLine(leftstring)
        Console.ReadLine()
    End Sub

End Module
```

Gives an output of **"Thi"** (the first 3 letters of the text string)


file:///C:/Documents and Settings/young/Local Settings/Application Data/Temporary Project...

```
Thi
```

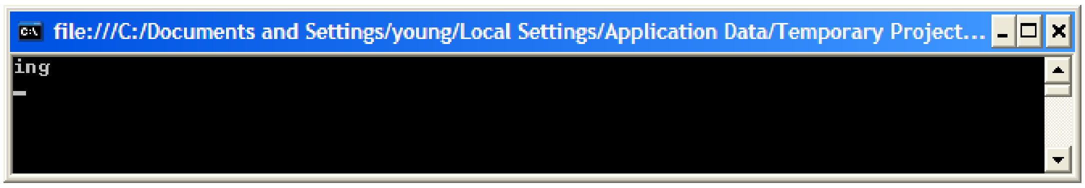
Using Right()

The Right function returns a specified number of characters from the right side of a string.

Tip: Use the Len function to find the number of characters in a string.

```
Module Module1

    Sub Main()
        Dim textstring As String
        Dim rightstring As String

        textstring = "This is a text string"
        rightstring = Right(textstring, 3)

        Console.WriteLine(rightstring)
        Console.ReadLine()
    End Sub

End Module
```

Gives an output of **"ing"** (the last 3 letters of the text string)



file:///C:/Documents and Settings/young/Local Settings/Application Data/Temporary Project...

```
ing
_
```

Using Mid()

The Mid function returns a specified number of characters from a string beginning from a given starting point.

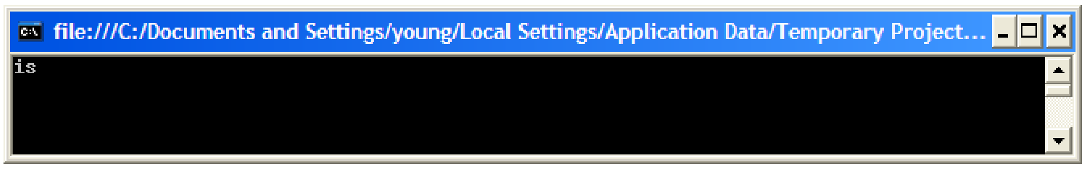
Tip: Use the Len function to determine the number of characters in a string.

```
Mid(string,start[,length])
```

Parameter	Description
string	Required. The string expression from which characters are returned
start	Required. Specifies the starting position. If set to greater than the number of characters in string, it returns an empty string ("")
length	Optional. The number of characters to return

```
Module Module1

    Sub Main()
        Dim textstring As String
        Dim midstring As String

        textstring = "This is a text string"
        midstring = Mid(textstring, 6, 2)

        Console.WriteLine(midstring)
        Console.ReadLine()
    End Sub

End Module
```

Would display “**is**”


file:///C:/Documents and Settings/young/Local Settings/Application Data/Temporary Project...

```
is
```

Trimming

The LTrim function removes spaces on the left side of a string.

```
LTrim(string)
```

Parameter	Description
string	Required. A string expression

The RTrim function removes spaces on the right side of a string.
Tip: Also look at the LTrim and the Trim functions.

```
RTrim(string)
```

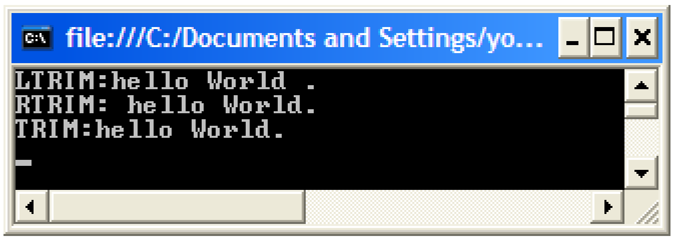
Parameter	Description
string	Required. A string expression

The Trim function removes spaces on both sides of a string.
Tip: Also look at the LTrim and the RTrim functions.

```
Trim(string)
```

Parameter	Description
string	Required. A string expression

```
Module Module1
    Sub Main()
        'Convert STRING to Single
        Dim stringtext As String
        Dim trimmedtext As String

        'Note space at the end and beginning of the string
        stringtext = " hello World "

        trimmedtext = LTrim(stringtext)
        Console.WriteLine("LTRIM:" & trimmedtext & ".")

        trimmedtext = RTrim(stringtext)
        Console.WriteLine("RTRIM:" & trimmedtext & ".")

        trimmedtext = Trim(stringtext)
        Console.WriteLine("TRIM:" & trimmedtext & ".")
        Console.ReadLine()
    End Sub
End Module
```



```
LTRIM:hello World .
RTRIM: hello World.
TRIM:hello World.
_
```

Converting Strings to Numbers

If the numerical data is stored as a string variable, it needs to be converted into a number before any calculations are performed upon it.

Sting to Integer

Converts a string to an integer (whole number)

```
Module Module1

    Sub Main()
        'Convert STRING to INTEGER
        Dim stringNumber As String
        Dim number As Integer

        stringNumber = "47"
        number = Convert.ToInt16(stringNumber)

        Console.WriteLine(number)
        Console.ReadLine()
    End Sub

End Module
```

String to Decimal

Converts a decimal number stored in a string variable to a real single precision number

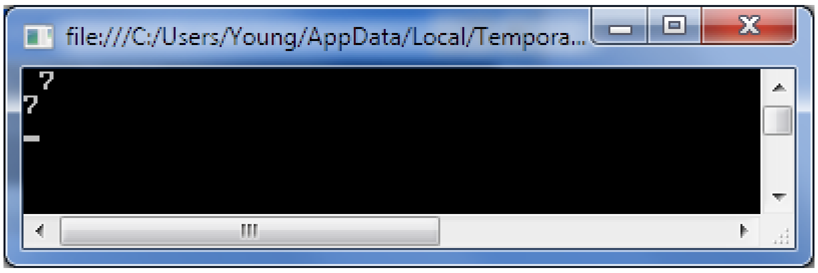
```
Module Module1

    Sub Main()
        'Convert STRING to Single
        Dim stringNumber As String
        Dim number As Single

        stringNumber = "53.66"
        number = Convert.ToSingle(stringNumber)

        Console.WriteLine(number)
        Console.ReadLine()
    End Sub

End Module
```

Converting Numbers to Strings

Use Str$ or CStr to convert a number into it's string representation. The difference between Str$ and CStr is that Str$ will add a leading space if the number is positive.

Integer to String

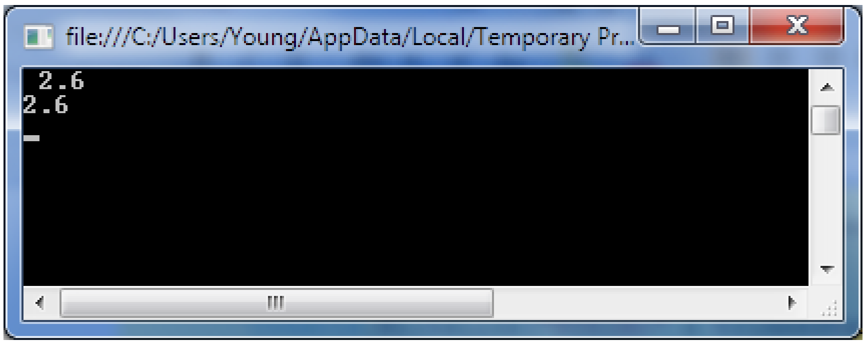
```
    Sub Main()
        Dim num As Integer
        num = 7

        Console.WriteLine(Str$(num))
        Console.WriteLine(CStr(num))
        Console.ReadLine()
    End Sub
```



```
 7
7
_
```

Single/Double to String

```
    Sub Main()
        Dim num As Double
        num = 2.6

        Console.WriteLine(Str$(num))
        Console.WriteLine(CStr(num))
        Console.ReadLine()
    End Sub
```



```
 2.6
2.6
_
```

Converting to and from ASCII

Use Asc to get a character's ASCII code. Use Chr to convert an ASCII code into a character.

Converting a character to ASCII

```
    Sub Main()
        Dim letter As Char
        Dim ASCII As Integer

        Console.WriteLine("Enter a character")
        letter = Console.ReadLine

        ASCII = Asc(letter)

        Console.WriteLine(ASCII)
        Console.ReadLine()
    End Sub
```

Converting an ASCII code to a character

```
    Sub Main()
        Dim letter As Char
        Dim ASCII As Integer

        Console.WriteLine("Enter an ASCII code")
        ASCII = Console.ReadLine

        letter = Chr(ASCII)

        Console.WriteLine(letter)
        Console.ReadLine()
    End Sub
```

String to Date

Converts a date stored in a string variable into a date.

```
Module Module1

    Sub Main()
        'Convert STRING to Date
        Dim stringdate As String
        Dim mydate As Date

        stringdate = "12/3/1978"
        mydate = Convert.ToDateTime(stringdate)

        Console.WriteLine(mydate)
        Console.ReadLine()
    End Sub

End Module
```

Changing Case

The UCase function converts a specified string to uppercase.

```
UCase(string)
```

Parameter	Description
string	Required. The string to be converted to uppercase

The LCase function converts a specified string to lowercase.

```
LCase(string)
```

Parameter	Description
string	Required. The string to be converted to lowercase

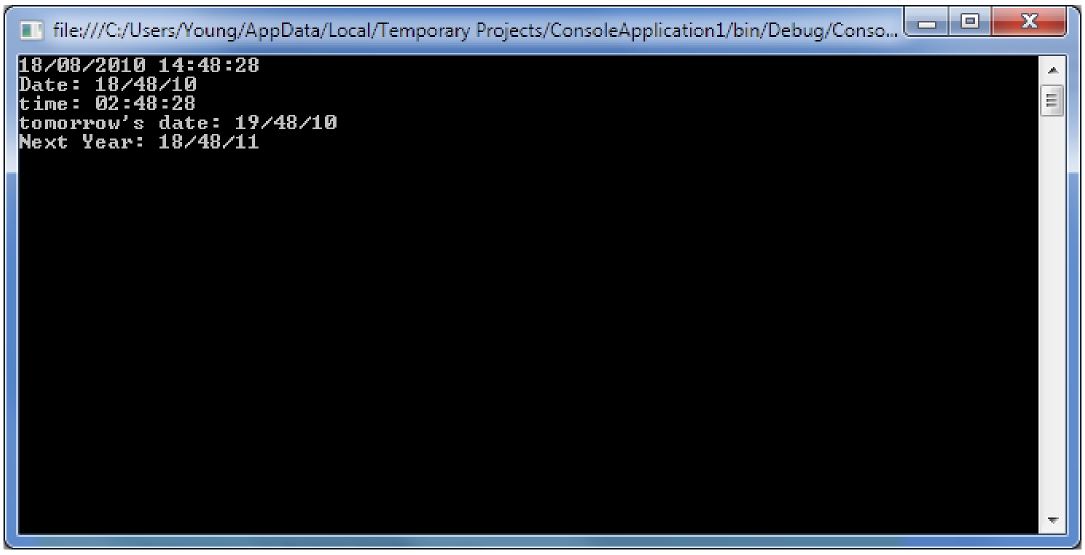
```
Module Module1

    Sub Main()
        'Convert STRING to Single
        Dim stringtext As String

        stringtext = "Some TeXt in MiXed cAsE"

        Console.WriteLine("As typed " & stringtext)
        Console.WriteLine("Lowercase " & LCase(stringtext))
        Console.WriteLine("Uppercase " & UCase(stringtext))

        Console.ReadLine()
    End Sub

End Module
```

Time and Date Manipulation

VB.net has a number of functions that help you manipulate dates and time

To get to the date and time use the `NOW()` function.

```
Imports System.Console
Module Module1

    Sub Main()
        Dim Today, tomorrow, nextyear As Date

        Today = Now()
        WriteLine(Now)
        WriteLine("Date: " & Format(Today, "dd/mm/yy"))
        WriteLine("time: " & Format(Today, "hh:mm:ss"))

        tomorrow = Today.AddDays(1)
        WriteLine("tomorrow's date: " & Format(tomorrow, "dd/mm/yy"))

        nextyear = Today.AddYears(1)
        WriteLine("Next Year: " & Format(nextyear, "dd/mm/yy"))

        ReadLine()
    End Sub

End Module
```


file:///C:/Users/Young/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Conso...

```
18/08/2010 14:48:28
Date: 18/48/10
time: 02:48:28
tomorrow's date: 19/48/10
Next Year: 18/48/11
```

VB.NET Split String

You may need to Split a String in VB.NET based on a character delimiter such as " "c.

```
    Sub Main()
        ' The file system path we need to split
        Dim s As String = "C:\Users\Sam\Documents\Perls\Main"

        ' Dim parts As String()
        Dim parts As String() = Split(s, "\")

        Console.WriteLine(parts(1))

        Console.ReadLine()
    End Sub
```

Simple Split call

The code below will split a String variable based on a space character, " "c. the results allocated to an array, New Char(), as well as a String() array to store the words in. Finally, we loop over the Strings and display them to the Console.

```
Module Module1

    Sub Main()
        Dim s As String = "there is a cat"

        Dim words As String() = s.Split(New Char() {" "c})

        ' Use For Each loop over words and display them
        Dim word As String
        For Each word In words
            Console.WriteLine(word)
        Next
    End Sub

End Module
```

The ( ) after the variable creates an array of an unknown number of elements.

=== Output of the program ===

```
there
is
a
cat
```

Splitting parts of file path

Here we see how you can Split a file system path into separate parts. We use a New Char() array with one string, "\""c, and then loop through and display the results.

```
Module Module1

    Sub Main()
        ' The file system path we need to split
        Dim s As String = "C:\Users\Sam\Documents\Perls\Main"

        ' Split the string on the backslash character
        Dim parts As String() = s.Split(New Char() {"\"c})

        ' Loop through result strings with For Each
        Dim part As String
        For Each part In parts
            Console.WriteLine(part)
        Next
    End Sub

End Module
```

=== Output of the program ===

```
C:
Users
Sam
Documents
Perls
Main
```

How to split based on words

Often you need to extract the words from a String or sentence in VB.NET. The code here needs to handle punctuation and non-word characters differently than the String Split method. Here we use Regex.Split to parse the words.

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        ' Declare iteration variable
        Dim s As String

        ' Loop through words in string
        Dim arr As String() = SplitWords("That is a cute cat, man!")

        ' Display each word. Note that punctuation is handled correctly.
        For Each s In arr
            Console.WriteLine(s)
        Next
        Console.ReadLine()
    End Sub

    Private Function SplitWords(ByVal s As String) As String()
        '
        ' Call Regex.Split function from the imported namespace.
        ' Return the result array.
        '
        Return Regex.Split(s, "\W+")
    End Function

End Module
```

=== Output of the program ===

```
That
is
a
cute
cat
```

manDescription of the example code. In the Main() subroutine you can see that two variables are declared. The second variable is a String() array that receives the results from the Private Function next.

Description of the Regex. The Function shown in the example calls the Regex.Split method, which can be accessed with dot notation, with no instance necessary. The second parameter to the method is a regular expression pattern.

Description of the Regex pattern. The pattern "\W+" is used, and this means "1 or more non-word characters". This pattern will match punctuation and spaces. Therefore, all those characters will be used as delimiters.

Splitting each line in a file
Here we see one way to Split each line in a file using File.ReadAllLines and Split. We have a comma-separated-values CSV file, and want to print out each value and its row number. Here is the input file "example.txt". Please see the below example.

The Split code example follows. It first reads in the file with ReadAllLines. This function puts each line in the file into an array element. The example next Splits on ","c. The final comment shows the output of the program.

=== Input file used ===

```
frontal,parietal,occipital,temporal
pulmonary artery,aorta,left ventricle
```

=== Example program that splits lines (VB.NET) ===

```
Imports System.IO

Module Module1
    Sub Main()
        Dim i As Integer = 0

        ' Loop through each line in array returned by ReadAllLines
        Dim line As String
        For Each line In File.ReadAllLines("example.txt")

            ' Split line on comma
            Dim parts As String() = line.Split(New Char() {","c})

            ' Loop over each string received
            Dim part As String
            For Each part In parts
                ' Display to Console
                Console.WriteLine("{0}:{1}", i, part)
            Next
            i += 1
        Next
    End Sub
End Module
```

=== Output of the program ===

```
0:frontal
0:parietal
0:occipital
0:temporal
1:pulmonary artery
1:aorta
1:left ventricle
```

Programming Projects

Write a program that reads in a string and displays the number of characters in the string.

Write a program that displays the ASCII code for any given character.

Write a program that will display the character for a given ASCII code.

Write a program that asks the user to type in a number with decimal places. The program should then display the rounded and the truncated number.

Write a program that asks the user for their surname and displays the surname in uppercase letters.

Write a program that displays today's date and formats the output appropriatly

rite a program that reads in a date, converts it into date format, adds a day and displays the next day's date.

Challenge projects

Write a program that takes two letters as input and displays all the letters of the alphabet between the two supplied letters (inclusive). For example, EJ produces EFGHIJ. The letters are to be printed in the order in which the specified letters are supplied. GB should produce GFEDCB. (**TIP**: Use the letters ASCII code – see page 53)

Write a program that asks the user for a word, and then displays the ASCII code for each letter in the word.

Hint: Process one character at a time, using the MID function.

Write a program that asks the user to enter a sentence, terminated by a full stop and the pressing of the Enter key. The program should count the number of words and display the result.

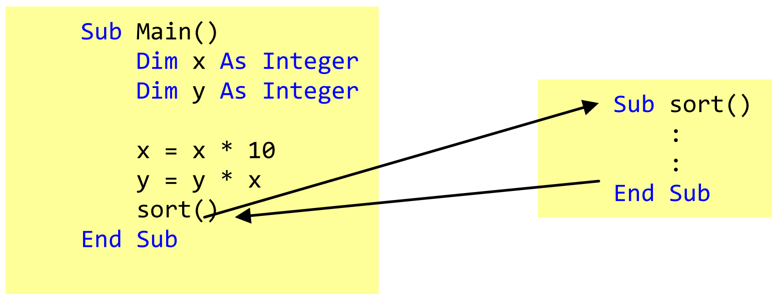
Hint: A word will end with a space or a full stop.

Write a palindrome tester. A palindrome is a word or sentence that reads the same backwards as forwards. The user should enter a string and the program should display whether the string is a palindrome or not.

12. Subroutines & Procedures

Initially, a program was written as one monolithic block of code. The program started at the first line of the program and continued to the end.

Program languages have now been developed to be structured. A problem can be divided into a number of smaller subroutines (also called procedures). From within one subroutine, another subroutine can be called and executed:



```
Sub Main()
    Dim x As Integer
    Dim y As Integer

    x = x * 10
    y = y * x
    sort()
End Sub
```

```
Sub sort()
    :
    :
End Sub
```

This gives the advantage of reusing common subroutines. It also makes it easier to program, as each procedure can be tested before moving onto the next (breaking a large problem into smaller ones).

With so many subroutines, the computer needs to know which one to execute first. In VB.Net, the `main()` subroutines is always executed first. When writing programs with lots of subroutines, it is good practice to create the subroutine before it is uses and the last subroutine is the very last in the module

```
Module Module1
    Dim num1 As Integer
    Dim num2 As Integer
    Dim answer As Integer

    Sub input_sub()
        Console.Clear()
        Console.WriteLine("Enter number 1")
        num1 = Console.ReadLine
        Console.WriteLine("Enter number 2")
        num2 = Console.ReadLine
    End Sub

    Sub Calculation()
        answer = num1 * num2
    End Sub

    Sub output_sub()
        Console.Write("the product of " & num1 & " and " & num2 & " is ")
        Console.WriteLine(answer)
        Console.ReadLine()
    End Sub

    Sub Main()
        input_sub()
        Calculation()
        output_sub()
    End Sub
End Module
```

13. Variable Scope

As described on page 7, a variable holds data while the program is running. The scope of a variable defines where it can be seen. They are classifies as either global or local

Global Variable

A global variable is declared in a module and is accessible from any procedure or function within that module.

Local Variables

A local variable is declared in a procedure or function and is only accessible within that procedure of function.

```
Module Module1
    Dim num1 As Integer
    Dim num2 As Integer
    Dim answer As Integer

    Sub input_sub()
        Console.Clear()
        Console.WriteLine("Enter number 1")
        num1 = Console.ReadLine
        Console.WriteLine("Enter number 2")
        num2 = Console.ReadLine
    End Sub

    Sub Calculation()
        answer = num1 * num2
    End Sub

    Sub output_sub()
        Console.Write("the product of " & num1 & " and " & num2 & "
is ")
        Console.WriteLine(answer)
    End Sub

    Sub Main()
        Dim answer As Char

        Do
            input_sub()
            Calculation()
            output_sub()

            Console.WriteLine("another go? Y/N")
            answer = Console.ReadLine()
        Loop Until UCase(answer) = "N"

    End Sub

End Module
```

Global variables, declared before any subroutines and are available throughout the

Local variable declared within a subroutine and is only available within this subroutine.

Scope of a Variable

The scope of a variable defines which subroutines can see and use a variable.

The sample code on the previous page defines the variables num1 and num2 as global; their scope is in subroutines:

- Input_sub
- Calculations
- Output_sub

However, the variable answer twice; once as a global variable and again as a local variable within main().

The scope of the integer version is available in:

- Input_sub
- Calculations

And the car version is available within

- Output_sub

If a global and a local variable share the same name, the local variable takes precedence.

Explicit and Strict

VB.Net does not require you to declare variables before you use them. This can lead to hours of wasted time as you hunt for an undeclared variable. Explicit and Strict allow you to only use declared variables. If you try to use a variable that has not been declared with a **DIM** statement, you will receive an error.

In earlier versions you may see the **explicit** command

```
Option Explicit On
Module Module1
   :
   :
   :
End Module
```

In VB.net 2005 onwards, you can also use **strict**

```
Option Strict On
Module Module1
   :
   :
   :
End Module
```

It is good practice to declare your variables before you use them, so always use the **strict on** when writing "real" programs

All identifiers must be declared before they can be used. The identifier can then be used in all procedure within the parent (see diagram on page **Error! Bookmark not defined.**). This is known as the scope.

If there is more than one declaration of an identifier with the same name, then the identifier used is the one declared within the **current procedure**.

Parameters

As mentioned above, local variables only have a lifespan of the procedure. Sometimes it is useful to pass a value from one procedure to another. This is done by using parameters (or arguments)

A parameter can be passed from one procedure to another by value or by reference.

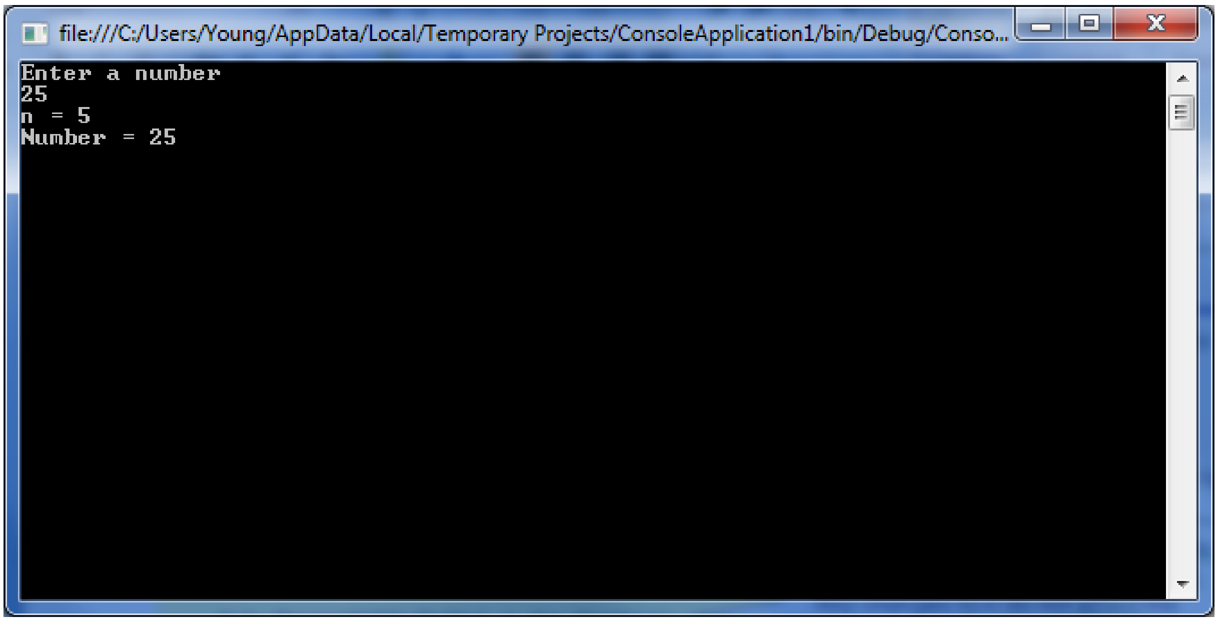
By Value

The word ByVal is short for "By Value". What it means is that you are passing a **copy** of a variable to your Subroutine. You can make changes to the copy and the original will not be altered.

```
Module Module1

    Sub WriteSQRT(ByVal n As Double)
        n = Math.Sqrt(n)
        Console.WriteLine("n = " & n)
    End Sub

    Sub Main()
        Dim number As Double
        Console.WriteLine("Enter a number")
        number = Console.ReadLine
        WriteSQRT(number)
        Console.WriteLine("Number = " & number)
        Console.ReadLine()
    End Sub

End Module
```

This procedure us expecting a double variable, which is known locally as **n**. Any changes to n do not effect the original variable

The variable **number** is passed to the subroutine *WriteSQRT*


file:///C:/Users/Young/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Conso...

```
Enter a number
25
n = 5
Number = 25
```

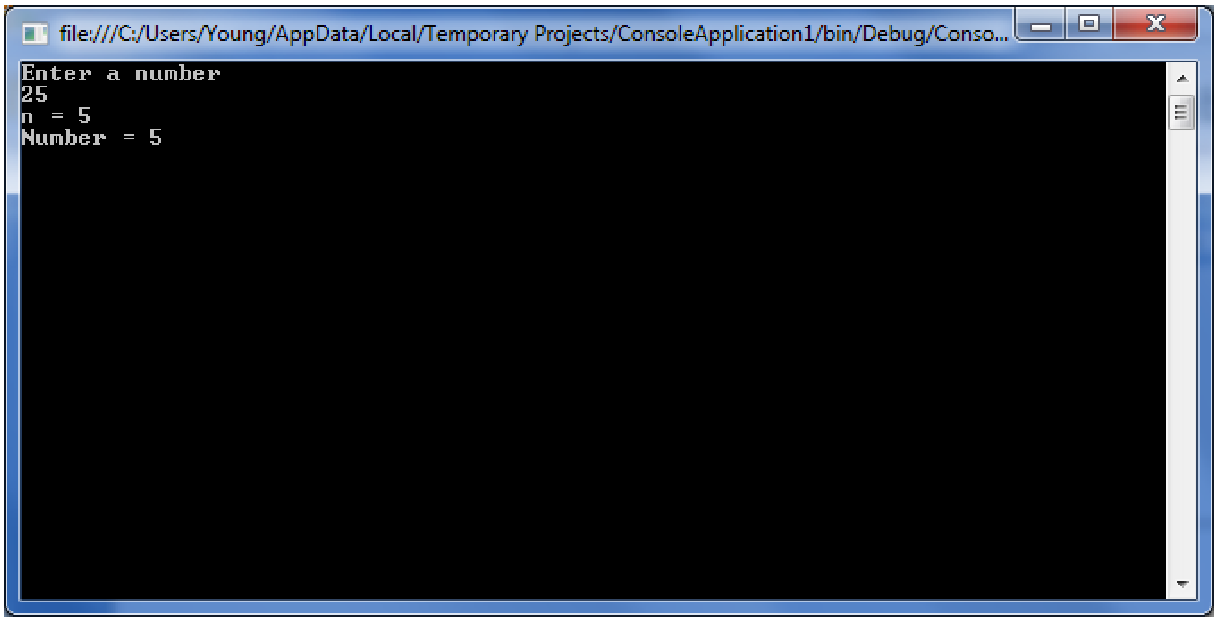
By Reference

ByRef is the alternative. This is short for By Reference. This means that you are not handing over a copy of the original variable but pointing to the original variable. Any change you make to the variable within your subroutine will effect the variable itself.

```
Module Module1

    Sub WriteSQRT(ByRef n As Double)
        n = Math.Sqrt(n)
        Console.WriteLine("n = " & n)
    End Sub

    Sub Main()
        Dim number As Double
        Console.WriteLine("Enter a number")
        number = Console.ReadLine
        WriteSQRT(number)
        Console.WriteLine("Number = " & number)
        Console.ReadLine()
    End Sub

End Module
```

This procedure us expecting a double variable, which is known locally as **n**. Any changes **WILL** effect the original variable

The variable **number** is passed to the subroutine *WriteSQRT*


file:///C:/Users/Young/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Conso...

```
Enter a number
25
n = 5
Number = 5
```

Programming Projects

Write and test a procedure Swap, which takes two integers as parameters and returns the first value passed to the procedure as the second value returned by the procedure and vice versa.

Write and test a procedure OutputSymbols, which takes two parameters: an integer n and a character symbol. The procedure is to display, on the same line, the symbol n times.

For example, the call OutputSymbols (5, ' # ') should display #####.

Write and test a procedure Sort, which takes two integers as parameters and returns them in ascending order.

For example, if No1 contained 5 and No2 contained 3, then the call Sort(No1, No2) will leave the value 3 in No1 and the value 5 in No2, but the call Sort(No2, No1) will leave the variable contents as they are.

Write a program to let the computer guess a number the user has thought of, within a range specified by you as the programmer.

Challenge Projects

The game 'Last one loses' is played by two players and uses a pile of n counters. Players take turns at removing 1, 2 or 3 counters from the pile. The game continues until there are no counters left and the winner is the one who does not take the last counter. Using procedures, write a program to allow the user to specify n in the range 10 — 50 inclusive and act as one player, playing at random until fewer than 5 counters remain. Try playing against your program, and then playing to win.

Create a procedure GetLotteryNumbers that will supply 6 unique random numbers between 1 and 49. One possible strategy, or algorithm, is:

```
Initialise an array by using a for loop to store the values 1 to 49
Repeatedly select a random element from array until a non-zero value is selected
Display this value
Set that element to zero
Repeat the above three steps until six numbers have been selected.
```

14. Functions

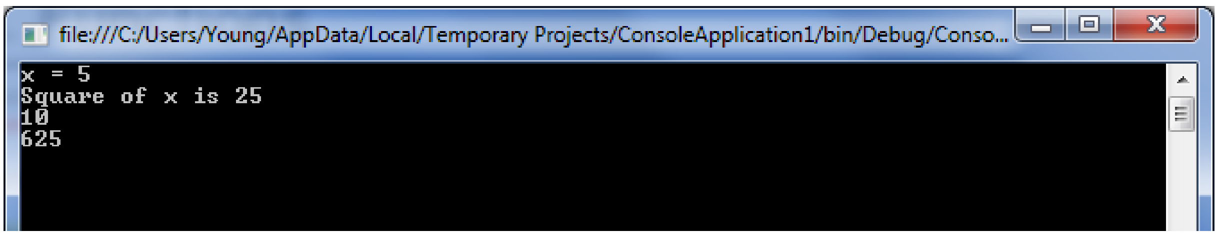
Functions are similar to subroutines, except that they always return a value. They are normally used in either **assignments** (A:=TaxA(370);) or **expressions** (IF taxA(15000) THEN….)

The function names doubles as a procedure name and a variable.

```
    Function square(ByVal x As Integer) As Integer
        square = x * x
    End Function
```

Square is the function name, that is expecting an integer to be passed(byref) to it.
The result is assigned to the function name which is dimensioned as an integer. The function name can be used as a variable containing the result within other procedures.

```
Module Module1

    Function square(ByVal x As Integer) As Integer
        square = x * x
    End Function

    Function sum(ByRef a As Integer, ByRef b As Integer) As Integer
        sum = a + b
    End Function

    Sub Main()
        Dim number As Double = 5

        Console.WriteLine("x = " & number)
        Console.WriteLine("Square of x is " & square(number))

        Console.WriteLine(sum(3, 7))
        Console.WriteLine(square(sum(16, 9)))

        Console.ReadLine()
    End Sub

End Module
```


file:///C:/Users/Young/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Conso...

```
x = 5
Square of x is 25
10
625
```

Programming languages, such as VB.net and spreadsheets, have many functions built-in. Examples include

SUM(range)	Spreadsheet: to add a block of cell values.
lcase(string)	VB: converts a string to upper case
ROUND(integer)	Round the integer up
RANDOM	Generate a random number

Programming Projects

⌨ Write a function to convert temperatures from Fahrenheit to Celsius. The function should take one integer parameter (the temperature in Fahrenheit) and return a real result (the temperature in Celsius). The formula for conversion is

Centigrade = (Fahrenheit — 32) * (5 / 9)

⌨ Write a function that converts a string passed as a parameter into a capitalised string. See page 46 for details on string manipulation)

⌨ Write a function that returns the total number of seconds, calculated from a whole number of hours, minutes and seconds provided as 3 parameters.

⌨ Write your own random function RandomNumber that returns values in the range from 1 to the integer supplied as parameter. Tip: use the Maths library (see page 18)

⌨ Write a tables tester. The program chooses 2 random numbers and asks the user what is the product of these 2 numbers. If the user gets the answer wrong, the correct answer should be displayed. The program should ask 10 questions and then display a score out of 10 and a suitable message.

15. Error Handling

When your program encounters an unexpected situation that prevents execution from taking place, for example a division by zero, a shortage of memory or an invalid parameter passed to a function, an error is raised. The program is interrupted and the following actions take place :

- The Number and Description fields of the Err object are initialized respectively with the number and description of the error.
- The call stack steps back function by function until it reaches the first active error handler.
- If an error handler is found, control is transferred to its first line. If not, control is returned to the system which displays a dialog box containing a description of the error.

The following code generates an error by dividing a number by 0. The error is detected and the program jumps to the errorhandler. A special object, called ERR contains the error number, description and other useful information.

```
Module Module1

    Sub Main()

        Dim a As Integer = 4
        Dim b As Integer = 0

        On Error GoTo errorhandler

        a = a / b

        Exit Sub

errorhandler:
        Console.WriteLine("An error has occured")
        Console.WriteLine(Err.Description)
        Console.ReadLine()

    End Sub

End Module
```

If you want VB.net to ignore errors (a very unwise thing to do) you can use the

On Error Resume Next

Which states that if an error is encountered, VB.net ignore it and goes onto the next statement.

```
Module Module1

    Sub Main()

        Dim a As Integer = 4
        Dim b As Integer = 0

        On Error Resume Next

        a = a / b

        Console.WriteLine("last line of the program")
        Console.ReadLine()

    End Sub

End Module
```

16. Using Text Files

Accessing special Folders

Locations of files can vary from machine to machine or user to user. The exact location of my Documents folder changes depending on who has logged on.

VB.net uses special system variables to hold the current users file locations, such as my documents, desktop, My Music, etc.

To get access the these variables, you must import the system.environment library.

NOTE: Not all locations are available due to system security

```
Option Explicit On
Imports System.Environment

Module Module1
    Dim mydocs As String
    Dim mymusic As String
    Dim myfavorites As String

    Sub main()

        mydocs = GetFolderPath(SpecialFolder.MyDocuments)
        mymusic = GetFolderPath(SpecialFolder.MyMusic)
        myfavorites = GetFolderPath(SpecialFolder.Favorites)

        Console.WriteLine(mydocs)
        Console.WriteLine(mymusic)
        Console.WriteLine(myfavorites)
        Console.ReadLine()

    End Sub
End Module
```

Using folders

To access sub-directories, concatenate the system folder path with the folder path and/or file name:

```
Option Explicit On
Imports System.Environment

Module Module1
    Dim mydocs As String
    Dim myfiles As String

    Sub main()

        mydocs = GetFolderPath(SpecialFolder.MyDocuments)
        myfiles = mydocs & "\textfiles"

        Console.WriteLine(myfiles)
        Console.ReadLine()
    End Sub
End Module
```

Files using Channels

The FILEOPEN command opens a file for input or output. It used the concept of having a filenumber to link the program to the physical file.

Reading Files (Input)

```
        FileOpen(1, "MyFile.txt", OpenMode.Input)
```

Filenumber

File name. This could include the filepath.

Open for input

Reading a line of text

To read a line of text from the opened file

```
DIM LineFromFile as string

LineFromFile = LineInput(1)
```

Read the line from filenumber 1

Closing file

```
FileClose(1)
```

Close filenumber 1

Writing a line of Text

```
        FileOpen(1, "MyFile.txt", OpenMode.output)
```

Filenumber

File name. This could include the filepath.

Open for output

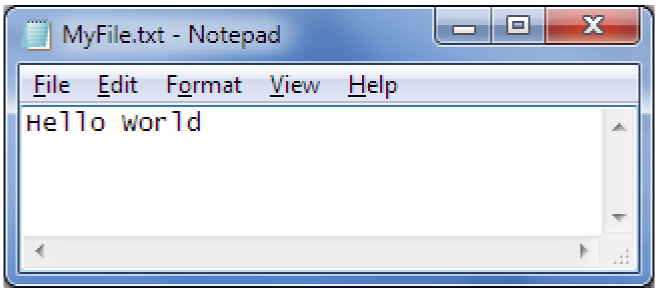
Printing a line of text

The PrintLine writes a string to a text file opened with a filenumber.

```
        Dim LineofText As String
        LineofText = "Hello World"
        PrintLine(1, LineofText)
```

Print to filenumber 1

The above code will produce the following text file:



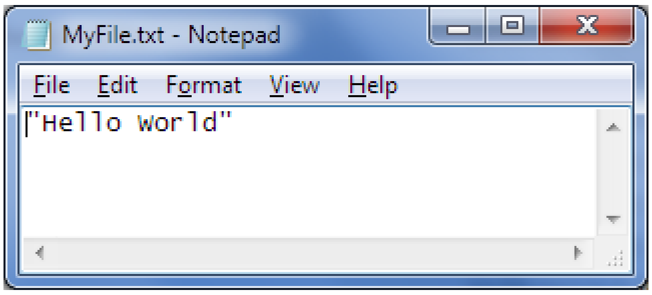
```
Hello World
```

Writing a line of text

The Writeline writes to a textfile opened with a filenumber BUT the string is enclosed in quotes

```
        Dim LineofText As String
        LineofText = "Hello World"
        Writeline(1, LineofText)
```

write to filenumber 1



```
"Hello World"
```

Creating CSV files with WRITELINE

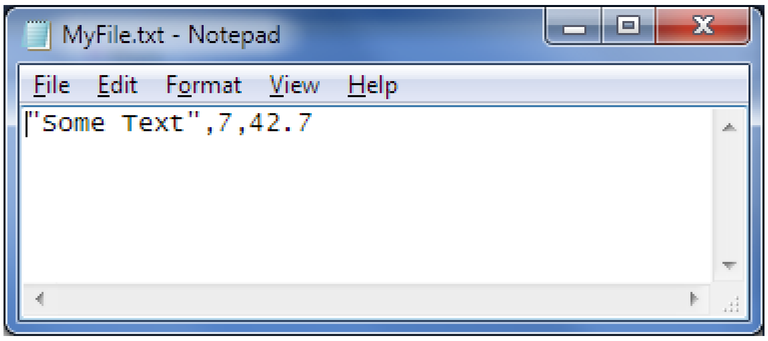
The comma-separated values (CSV) file format is a file formats used to store tabular data in which numbers and text are stored in plain textual form that can be read in a text editor, spreadsheet or Database.

Lines in the text file represent rows of a table, and commas in a line separate what are fields in the tables row.

The following example used the WriteLine statement to create a CSV file with 3 variables:

If writeline displays the contents on the console rather than writing out to file, use filesystem.writeline. This forces the output to the file.

```
    Sub Main()
        Dim Field1 As String
        Dim Field2 As Integer
        Dim field3 As Double

        Field1 = "Some Text"
        field2 = 7
        field3 = 42.7

        FileOpen(1, "S:\MyFile.txt", OpenMode.Output)
        Filesystem.WriteLine(1, Field1, field2, field3)
        FileClose(1)
    End Sub
```



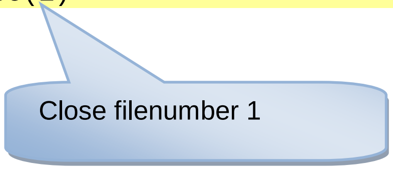
```
"Some Text",7,42.7
```

NOTE: Strings are enclosed in quotes, numbers are not enclosed in quotes

For other ways of manipulating CSV files, see page 82

Closing file

```
FileClose(1)
```



StreamWriter with text files

Two objects StreamReader and StreamWriter are used to read and write data in a text file.

Both of these commands are stored in the System.IO library, so you will need to import it into your programme. The following line needs to be added B System.IO by adding **before** the Module definition

```
Imports System.IO
```

```
Option Explicit On
Imports System.IO
Imports System.Environment

Module Write

'create a variable to write a stream of characters to a text file
    Dim CurrentFileWriter As StreamWriter

    Sub Main()
        Dim FileName, TextString As String
        Dim Count As Integer


        FileName = GetFolderPath(SpecialFolder.MyDocuments) & “text.txt”
        CurrentFileWriter = New StreamWriter(FileName)
        Console.WriteLine("File being created")
        CurrentFileWriter.WriteLine("File ceated on " & Now())

        For Count = 1 To 5
            TextString = Rnd() * 100
            Console.WriteLine("Random number " & Count & " is " &
TextString)
            CurrentFileWriter.WriteLine("Random number " & Count & " is " &
TextString)
        Next

        CurrentFileWriter.Close() ' close file
        Console.WriteLine("File saved")
        Console.ReadLine()

    End Sub

End Module
```

StreamReader with text files

The StreamReader can either read the contents of the whole file into a variable, or read one line at a time.

.ReadToEnd reads the entire file into a variable
.ReadLine reads a single line (up to the CR code)

```
Option Explicit On used
Imports System.IO
Imports System.Environment

Module Module1
    Dim CurrentFileReader As StreamReader
    Sub Main()
        Dim FileName, TextString As String

        TextString = ""
        FileName = GetFolderPath(SpecialFolder.MyDocuments) & “text.txt”
        CurrentFileReader = New StreamReader(FileName) 'opens the file

        If File.Exists(FileName) Then
            TextString = CurrentFileReader.ReadToEnd
        Else
            Console.WriteLine("File does not exist")
        End If
        CurrentFileReader.Close() ' close file
        Console.WriteLine(TextString)
        Console.ReadLine()

    End Sub

End Module
```

Reading lines from a Text File

Add examples

17. Binary Files

Writing data as strings of text files is OK, but it would be more useful to write data structures (such as records) to a file.

If you have create your own data type (see page 42) you can write these data types to a binary file. The program knows the structure of the data type and writes records to the file or reads them in from file.

With a binary file, you can:

- Create a binary file (see page 79)
- Write user data types to a file (see page 80)
- Append (add to the end) of a binary file (see page 81)
- Read a file of data types from file (see page 80)

All the commands are stored in the System.IO library, so you will need to import it into your programme. The following line needs to be added B System.IO by adding **before** the Module definition

```
Imports System.IO
```

Following examples this user defined data type:

```
    Structure TBook
        Dim ISBN As String
        Dim Title As String
        Dim Price As Decimal
        Dim YearOfPublication As Integer
    End Structure
    Dim books(10) As TBook
```

Creating Binary Files

```
    Sub CreateNewbookFile()
        Dim CurrentFile As FileStream
        Dim answer As Char

        If File.Exists(filename) Then
            Console.WriteLine("File already exists. Are you sure
you want to create a new file? Y/N")
            answer = Console.ReadLine().ToUpper

            If answer = "Y" Then
                CurrentFile = New FileStream(filename,
FileMode.Create)
                Console.WriteLine("File created")
            End If
        Else
            CurrentFile = New FileStream(filename,
FileMode.CreateNew)
            Console.WriteLine("File created")
        End If
        Console.ReadLine()
    End Sub
```

Writing Binary Files

```
    Sub writebooks()
        Dim CurrentFile As FileStream
        Dim CurrentFileWriter As BinaryWriter
        Dim index As Integer

        CurrentFile = New FileStream(FileName, FileMode.Create)
        CurrentFileWriter = New BinaryWriter(CurrentFile)

        For index = 0 To BookCount
            CurrentFileWriter.Write(books(index).ISBN)
            CurrentFileWriter.Write(books(index).title)
            CurrentFileWriter.Write(books(index).price)
            CurrentFileWriter.Write(books(index).YearOfPublication)
        Next
        CurrentFile.Close()
        CurrentFileWriter.Close()
        Console.WriteLine("Books written to file")
        Console.ReadLine()
    End Sub
```

Reading Binary Files

```
    Sub Readbooks()
        Dim CurrentFile As FileStream
        Dim CurrentFileReader As BinaryReader
        Dim index As Integer

        CurrentFile = New FileStream(filename, FileMode.Open)
        CurrentFileReader = New BinaryReader(CurrentFile)

        index = 0

        Do While CurrentFile.Position < CurrentFile.Length
            books(index).ISBN = CurrentfileReader.readstring
            books(index).title = CurrentFileReader.ReadString
            books(index).price = CurrentFileReader.ReadDecimal
            books(index).YearOfPublication =
CurrentFileReader.ReadInt32
            index = index + 1
        Loop
        BookCount = index - 1
        CurrentFile.Close()
        CurrentFileReader.Close()
        Console.WriteLine(index & "Books read from file")
        Console.ReadLine()
    End Sub
```

Appending to a Binary File

```
   Sub AppendBook()
        Dim CurrentFile As FileStream
        Dim CurrentFileWriter As BinaryWriter
        Dim index As Integer
        Dim answer As Char

        CurrentFile = New FileStream(filename, FileMode.Append)
        CurrentFileWriter = New BinaryWriter(CurrentFile)

        Do
            Console.WriteLine("Enter the book's details")
            Console.Write("ISBN :")
            books(index).ISBN = Console.ReadLine()
            Console.Write("Title :")
            books(index).title = Console.ReadLine()
            Console.Write("Price :")
            books(index).price = Console.ReadLine()
            Console.Write("Year of Publication :")
            books(index).YearOfPublication = Console.ReadLine()

            CurrentFileWriter.Write(books(index).ISBN)
            CurrentFileWriter.Write(books(index).title)
            CurrentFileWriter.Write(books(index).price)
            CurrentFileWriter.Write(books(index).YearOfPublication)

            Console.WriteLine("Add another book? Y/N")
            answer = Console.ReadLine().ToUpper
            index = index + 1

        Loop Until (answer = "N" Or index > 10)

        BookCount = index - 1

        CurrentFile.Close()
        CurrentFileWriter.Close()
        Console.WriteLine("Books written to file")
        Console.ReadLine()
    End Sub
```

18. CSV Files

```
Module Module1

    Sub Main()
        Dim textstring As String
        Dim midstring As String

        textstring = "This is a text string"
        midstring = Mid(textstring, 6, 2)

        Console.WriteLine(midstring)
        Console.ReadLine()
    End Sub


    Sub ReadCSVFileToArray()
        Dim strfilename As String
        Dim num_rows As Long
        Dim num_cols As Long
        Dim x As Integer
        Dim y As Integer
        Dim strarray(1, 1) As String

        ' Load the file.
        strfilename = "test.csv"

        'Check if file exist
        If File.Exists(strfilename) Then
            Dim tmpstream As StreamReader = File.OpenText(strfilename)
            Dim strlines() As String
            Dim strline() As String

            'Load content of file to strLines array
            strlines = tmpstream.ReadToEnd().Split(Environment.NewLine)

            ' Redimension the array.
            num_rows = UBound(strlines)
            strline = strlines(0).Split(",")
            num_cols = UBound(strline)
            ReDim strarray(num_rows, num_cols)

            ' Copy the data into the array.
            For x = 0 To num_rows
                strline = strlines(x).Split(",")
                For y = 0 To num_cols
                    strarray(x, y) = strline(y)
                Next
            Next


        End If

    End Sub



End Module
```

19. Summary of VB Functions

Date/Time Functions

Function	Description
CDate	Converts a valid date and time expression to the variant of subtype Date
Date	Returns the current system date
DateAdd	Returns a date to which a specified time interval has been added
DateDiff	Returns the number of intervals between two dates
DatePart	Returns the specified part of a given date
DateSerial	Returns the date for a specified year, month, and day
DateValue	Returns a date
Day	Returns a number that represents the day of the month (between 1 and 31, inclusive)
FormatDateTime	Returns an expression formatted as a date or time
Hour	Returns a number that represents the hour of the day (between 0 and 23, inclusive)
IsDate	Returns a Boolean value that indicates if the evaluated expression can be converted to a date
Minute	Returns a number that represents the minute of the hour (between 0 and 59, inclusive)
Month	Returns a number that represents the month of the year (between 1 and 12, inclusive)
MonthName	Returns the name of a specified month
Now	Returns the current system date and time
Second	Returns a number that represents the second of the minute (between 0 and 59, inclusive)
Time	Returns the current system time
Timer	Returns the number of seconds since 12:00 AM
TimeSerial	Returns the time for a specific hour, minute, and second
TimeValue	Returns a time
Weekday	Returns a number that represents the day of the week (between 1 and 7, inclusive)
WeekdayName	Returns the weekday name of a specified day of the week
Year	Returns a number that represents the year

Conversion Functions

Function	Description
Asc	Converts the first letter in a string to ANSI code
CBool	Converts an expression to a variant of subtype Boolean
CByte	Converts an expression to a variant of subtype Byte
CCur	Converts an expression to a variant of subtype Currency
CDate	Converts a valid date and time expression to the variant of subtype Date
CDbl	Converts an expression to a variant of subtype Double
Chr	Converts the specified ANSI code to a character
CInt	Converts an expression to a variant of subtype Integer
CLng	Converts an expression to a variant of subtype Long
CSng	Converts an expression to a variant of subtype Single
CStr	Converts an expression to a variant of subtype String
Hex	Returns the hexadecimal value of a specified number
Oct	Returns the octal value of a specified number

Format Functions

Function	Description
FormatCurrency	Returns an expression formatted as a currency value
FormatDateTime	Returns an expression formatted as a date or time
FormatNumber	Returns an expression formatted as a number
FormatPercent	Returns an expression formatted as a percentage

Math Functions

Function	Description
Abs	Returns the absolute value of a specified number
Atn	Returns the arctangent of a specified number
Cos	Returns the cosine of a specified number (angle)
Exp	Returns *e* raised to a power
Hex	Returns the hexadecimal value of a specified number
Int	Returns the integer part of a specified number
Fix	Returns the integer part of a specified number
Log	Returns the natural logarithm of a specified number
Oct	Returns the octal value of a specified number
Rnd	Returns a random number less than 1 but greater or equal to 0
Sgn	Returns an integer that indicates the sign of a specified number
Sin	Returns the sine of a specified number (angle)
Sqr	Returns the square root of a specified number
Tan	Returns the tangent of a specified number (angle)

Array Functions

Function	Description
Array	Returns a variant containing an array
Filter	Returns a zero-based array that contains a subset of a string array based on a filter criteria
IsArray	Returns a Boolean value that indicates whether a specified variable is an array
Join	Returns a string that consists of a number of substrings in an array
LBound	Returns the smallest subscript for the indicated dimension of an array
Split	Returns a zero-based, one-dimensional array that contains a specified number of substrings
UBound	Returns the largest subscript for the indicated dimension of an array

String Functions

Function	Description
InStr	Returns the position of the first occurrence of one string within another. The search begins at the first character of the string
InStrRev	Returns the position of the first occurrence of one string within another. The search begins at the last character of the string
LCase	Converts a specified string to lowercase
Left	Returns a specified number of characters from the left side of a string
Len	Returns the number of characters in a string
LTrim	Removes spaces on the left side of a string
RTrim	Removes spaces on the right side of a string
Trim	Removes spaces on both the left and the right side of a string
Mid	Returns a specified number of characters from a string
Replace	Replaces a specified part of a string with another string a specified number of times
Right	Returns a specified number of characters from the right side of a string
Space	Returns a string that consists of a specified number of spaces
StrComp	Compares two strings and returns a value that represents the result of the comparison
String	Returns a string that contains a repeating character of a specified length
StrReverse	Reverses a string
UCase	Converts a specified string to uppercase

Other Functions

Function	Description
CreateObject	Creates an object of a specified type
Eval	Evaluates an expression and returns the result
GetLocale	Returns the current locale ID
GetObject	Returns a reference to an automation object from a file
GetRef	Allows you to connect a VBScript procedure to a DHTML event on your pages
InputBox	Displays a dialog box, where the user can write some input and/or click on a button, and returns the contents
IsEmpty	Returns a Boolean value that indicates whether a specified variable has been initialized or not
IsNull	Returns a Boolean value that indicates whether a specified expression contains no valid data (Null)
IsNumeric	Returns a Boolean value that indicates whether a specified expression can be evaluated as a number
IsObject	Returns a Boolean value that indicates whether the specified expression is an automation object
LoadPicture	Returns a picture object. Available only on 32-bit platforms
MsgBox	Displays a message box, waits for the user to click a button, and returns a value that indicates which button the user clicked
RGB	Returns a number that represents an RGB color value
Round	Rounds a number
ScriptEngine	Returns the scripting language in use
ScriptEngineBuildVersion	Returns the build version number of the scripting engine in use
ScriptEngineMajorVersion	Returns the major version number of the scripting engine in use
ScriptEngineMinorVersion	Returns the minor version number of the scripting engine in use
SetLocale	Sets the locale ID and returns the previous locale ID
TypeName	Returns the subtype of a specified variable
VarType	Returns a value that indicates the subtype of a specified variable

Formatting Symbols

The Format function converts a value to a text string and gives you control over the string's appearance. For example, you can specify the number of decimal places for a numeric value, leading or trailing zeros, currency formats, and portions of the date.

Character	Description
None No formatting	Display the number with no formatting.
:	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings.
/	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings.
C	Display the date as ddddd and display the time as t t t t t, in that order. Display only date information if there is no fractional part to the date serial number; display only time information if there is no integer portion.
D	Display the day as a number without a leading zero (1 - 31).
dd	Display the day as a number with a leading zero (01 - 31).
ddd	Display the day as an abbreviation (Sun - Sat).
dddd	Display the day as a full name (Sunday - Saturday).
ddddd	Display the date as a complete date (including day, month, and year), formatted according to your system's short date format setting. The default short date format is m/d/yy.
dddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the long date setting recognized by your system. The default long date format is mmmm dd, yyyy.
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday).
ww	Display the week of the year as a number (1 - 53).
m	Display the month as a number without a leading zero (1 - 12). If m immediately follows h or hh, the minute rather than the month is displayed.
MM	Display the month as a number with a leading zero (01 - 12). If m immediately follows h or hh, the minute rather than the month is displayed.
MMM	Display the month as an abbreviation (Jan - Dec).
MMMM	Display the month as a full month name (January - December).
q	Display the quarter of the year as a number (1 - 4).
y	Display the day of the year as a number (1 - 366).
yy	Display the year as a 2-digit number (00 - 99).

yyyy	Display the year as a 4-digit number (100 - 9666).
h	Display the hour as a number without leading zeros (0 - 23).
hh	Display the hour as a number with leading zeros (00 - 23).
n	Display the minute as a number without leading zeros (0 - 59).
nn	Display the minute as a number with leading zeros (00 - 59).
s	Display the second as a number without leading zeros (0 - 59).
ss	Display the second as a number with leading zeros (00 - 59).
t t t t t	Display a time as a complete time (including hour, minute, and second), formatted using the time separator defined by the time format recognized by your system. A leading zero is displayed if the leading zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is h:mm:ss.
AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 P.M.
am/pm	Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 P.M.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 P.M.
a/p	Use the 12-hour clock and display a lowercase A with any hour before noon; display a lowercase P with any hour between noon and 11:59 P.M.
AMPM	Use the 12-hour clock and display the AM string literal as defined by your system with any hour before noon; display the PM string literal as defined by your system with any hour between noon and 11:59 P.M. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as defined by your system settings. The default format is AM/PM.
0 Digit placeholder	Display a digit or a zero. If the expression has a digit in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position. If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, display leading or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, round the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, display the extra digits without modification.
# Digit placeholder	Display a digit or nothing. If the expression has a digit in the position where the # appears in the format string, display it; otherwise, display nothing in that position. This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are # characters on either side of the decimal separator in the format expression.
. Decimal placeholder	In some locales, a comma is used as the decimal separator. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers

	smaller than 1 begin with a decimal separator. If you always want a leading zero displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator instead. The actual character used as a decimal placeholder in the formatted output depends on the Number Format recognized by your system.
% Percent placeholder	The expression is multiplied by 100. The percent character (%) is inserted in the position where it appears in the format string.
, Thousand separator	In some locales, a period is used as a thousand separator. The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a thousand separator surrounded by digit placeholders (0 or #). Two adjacent thousand separators or a thousand separator immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed." You can scale large numbers using this technique. For example, you can use the format string "##0,," to represent 100 million as 100. Numbers smaller than 1 million are displayed as 0. Two adjacent thousand separators in any position other than immediately to the left of the decimal separator are treated simply as specifying the use of a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format recognized by your system.
: Time separator	In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings.
/ Date separator	In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings.
E- E+ e- e+ Scientific format	If the format expression contains at least one digit placeholder (0 or #) to the right of E-, E+, e-, or e+, the number is displayed in scientific format and E or e is inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a minus sign next to negative exponents and a plus sign next to positive exponents.
- + $ ( ) space Display a literal character	To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" ").
\\ Display the next character in the format string	Many characters in the format expression have a special meaning and can't be displayed as literal characters unless they are preceded by a backslash. The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\\). Examples of characters that can't be displayed as literal characters are the date- and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, and /:), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &, <, >, and !).

"ABC" Display the string inside the double quotation marks	To include a string in format from within code, you must use Chr(34) to enclose the text (34 is the character code for a double quotation mark).
@ Character placeholder	Display a character or a space. If the string has a character in the position where the @ appears in the format string, display it; otherwise, display a space in that position. Placeholders are filled from right to left unless there is an ! character in the format string. See below.
& Character placeholder	Display a character or nothing. If the string has a character in the position where the & appears, display it; otherwise, display nothing. Placeholders are filled from right to left unless there is an ! character in the format string. See below.
< Force lowercase	Display all characters in lowercase format.
> Force uppercase	Display all characters in uppercase format.
! Force left to right fill of placeholders	The default is to fill from right to left.

Named Formats

Visual Basic provides several standard formats to use with the Format function. Instead of using symbols, you specify these formats by name in the format argument of the Format function. Always enclose the format name in double quotation marks (""). The following table lists the format names you can use.

Named format	Description
General Number	Shows numbers as entered.
Currency	Shows negative numbers inside parentheses.
Fixed	Shows at least one digit.
Standard	Uses a thousands separator.
Percent	Multiplies the value by 100 with a percent sign at the end.
Scientific	Uses standard scientific notation.
General Date	Shows date and time if expression contains both. If expression is only a date or a time, the missing information is not displayed.
Long Date	Uses the Long Date format specified in the Regional Settings dialog box of the Microsoft Windows Control Panel.
Medium Date	Uses the dd-mmm-yy format (for example, 03-Apr-93)
Short Date	Uses the Short Date format specified in the Regional Settings dialog box of the Windows Control Panel.
Long Time	Shows the hour, minute, second, and "AM" or "PM" using the h:mm:ss format.
Medium Time	Shows the hour, minute, and "AM" or "PM" using the "hh:mm AM/PM" format.
Short Time	Shows the hour and minute using the hh:mm format.

Yes/No	Any nonzero numeric value (usually - 1) is Yes. Zero is No.
True/False	Any nonzero numeric value (usually - 1) is True. Zero is False.
On/Off	Any nonzero numeric value (usually - 1) is On. Zero is Off.