INTRODUCTION TO VB.NET PROGRAMMING

VB.NET stands for Visual Basic .NET. Visual Basic is the new programming language that lets you program window GUI programs very fast. GUI stands for Graphic User Interface. All windows programs that you use are GUI's. VB.NET replaces VB6. If you know VB6 it will be easy for you to learn VB.NET. If you do not know VB6 or any programming language you will find, it will be quite a challenge to learn how to program using VB.NET. VB.NET is an improved programming language based on VB6. It is more object oriented and you may find it more difficult to use compared to VB6. All modern programming languages are now object orientated. Object oriented is a fancy term that lets you program with many different program groups called modules and classes. In this lesson we will deal with modules. Proceeding lessons will deal with objects and classes. Objects are defined by classes specifying data and programming statements. These guides will teach you VB.NET programming with step-by-step instructions. If you do all the exercises in these lessons you will become a very good VB.Net programmer. In each lesson we use bold to introduce new programming concepts, underline for important programming concepts, violet for program definitions and blue for programming code examples.

VB.NET Data Types and Variables

A program contains special words to instruct a computer what to do. The words form a programming statement, just like we use words to form a sentence. Which words are used defines the programming language. Each programming language has their own set of words. These sets of words are also called keywords. The words chosen for Visual Basic make it easy for some one to learn programming. Programs also contain values to represent quantities. The values are called data. Data values can be numeric data like 10.5 or string data like "hello". The different types of data are known as data types. Here is a chart of all data types used in Visual Basic .Net.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Sample value</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>&quot;hello there&quot;</td>
<td>Alphanumeric characters, digits and other characters</td>
</tr>
<tr>
<td>Char</td>
<td>'a'C</td>
<td>Single characters the C stands for character</td>
</tr>
<tr>
<td>Integer</td>
<td>-5</td>
<td>Larger numbers ranging from -2147483648 to 2147483647</td>
</tr>
<tr>
<td>Long</td>
<td>18459038</td>
<td>Very Large whole numbers</td>
</tr>
<tr>
<td>Boolean</td>
<td>True</td>
<td>True or False</td>
</tr>
<tr>
<td>Byte</td>
<td>10</td>
<td>small numbers range 0 to 255</td>
</tr>
<tr>
<td>Short</td>
<td>1234</td>
<td>medium numbers ranging from -32,768 to 32,767</td>
</tr>
<tr>
<td>Single</td>
<td>5.678</td>
<td>Single-precision floating point numbers with six digits of accuracy.</td>
</tr>
<tr>
<td>Double</td>
<td>-2.4585E30</td>
<td>Double-precision floating point numbers with 14 digits of accuracy</td>
</tr>
<tr>
<td>Decimal</td>
<td>10.50</td>
<td>Decimal fractions, such as dollars and cents.</td>
</tr>
<tr>
<td>Date</td>
<td>#5/23/2002#</td>
<td>month, day, year, various date formats available</td>
</tr>
<tr>
<td>Object</td>
<td>frm as Form</td>
<td>Objects represent a group of many values</td>
</tr>
</tbody>
</table>
Data is stored in computer memory. The location and value where the data is stored is represented by a **variable**. A variable gets an identification name like: \textit{x}. The identification name must start with a letter. Additional letters or digits may follow the first letter, like: \textit{x1}.

**VB .Net Program**

A VB.Net program is made up of variables and programming statements enclosed in a **module**. This is a big step for someone to understand. The simplest way to understand this is just to realize that variables and programming statements need to be grouped together by something. This something is called a **module**. Just like a neighborhood groups together homes and streets.

A VB.NET program may contain many **module** just like a city contains many neighborhoods. We will now learn to use variables and programming statements and place them into a module to form a VB.Net program.

**Declaring Variables**

Before you can use a variable in a program it has to be **declared**. To declare a variable you use the **Dim** keyword and specify a data type list in the preceding data type table.

\[
\text{Dim x as Integer} \quad \text{Code (blue)} \\
\text{Dim identifier As datatype} \quad \text{Definition (violet)}
\]

In the above example \textit{x} is declared as a variable of **Integer** data type. **Dim** means declare variable, **as** means specify a **data type**.
Using Variables

Think of a variable as a box that holds a value represented by an identifier. If you know the name of the identifier then you can get its value. The value is stored in computer memory represented by the variable.

After you declare a variable, you can give it a value. When you give a value to a variable it is known as assigning or initializing. Here is an example declaring an Integer variable \( x \) and initializing to an integer value.

\[
\text{Dim } x \text{ As Integer} \\
\text{ } \\
\text{x = 5}
\]

Here is an example of declaring a String variable and initializing it with a string value:

\[
\text{Dim name As String} \\
\text{name = "Ed"}
\]

Later in your program you can assign different values to your variables:

\[
\text{name = "Tom"}
\]

In this situation the old value is replaced with the new value. The old value disappear, it is gone forever.

To retrieve a value from a variable, you assign a variable to another variable shown as follows:

\[
\text{Dim x As Integer} \\
\text{Dim y As Integer} \\
\text{x = 10} \\
\text{y = x}
\]

In this situation the value of variable \( x \) is assigned to variable \( y \). Remember, it is the value represented by the variable that is assigned, not the name of the variable. The value is obtained using the name of a variable. As you use your variables the values will be always changing.

VB.Net also lets you declare and assign variables at the same time.

\[
\text{Dim x As Integer = 10} \\
\text{Dim name As String = "Ed"}
\]

You will find this very convenient
VB .Net Sample Program

We are now ready to write our first VB .Net program. In this sample program we will initialize some variables and print their values to the screen. Programming languages use programming statements to do things. A statement includes command words called keywords and data values. Our statement will be a statement that allows you to print out messages on the computer screen. Messages are String constants enclosed by " double quotes " like “hello”. This is a Console program, meaning the output is sent to the screen, and the user types in the input responses. A Console program is a little different than a GUI program. A GUI program lets the user interact with Buttons and Text Boxes.

In our program we first declare a name and age variable

    Dim name As String
    Dim age As Integer

We then initialize name to “Tom” and age to 20.

    name = "Tom"
    age = 20

Now we want to print the name and age to the screen. We use we use the Console.Write and Console.writeLine statements to do this.

Here we print   "Hi, my name is  Tom" message on the computer screen.

    Console.Write    "Hi, my name is 

You can also use a variable to print out values to the screen.

    Console.WriteLine (name)

Console is an Object that knows how to print messages and values to the screen and retrieves values from the keyboard. Write and WriteLine are called subs the let the object do things. We will teach you all about subs soon. The difference between Write and WriteLine is that WriteLine print ends with a new line and Write just prints out the message or value. We can continue to add more lines to our first program.

    Console.Write("I am now ")
    Console.Write(age)
    Console.WriteLine(" years old.")
    Console.Read()
We use `Console.Read()` to pause the program when it ends so the screen will not disappear when the program finishes.

Here is our example program. You will notice it is enclosed in a **Module** and a **Sub** called **Main**. The Sub encloses the programming statements. The Module encloses the Sub Main. A module can have many Subs. The Main Sub is the first sub to start executing in a module contained in a program.

```vbnet
Module Module1

    Sub Main()

        Dim name As String
        Dim age As Integer
        name = "Tom"
        age = 20
        Console.Write("Hi, my name is ")
        Console.WriteLine(name)
        Console.Write("I am now ")
        Console.Write(age)
        Console.WriteLine(" years old.")

        Console.Read() ' pause

    End Sub

End Module
```

Here are the step-by-step instructions to run the program.

Before you can write and run this program you first need to create a **new Project**. Make sure you have Visual Basic .Net installed on your computer. We are using Visual Basic .Net version 2003.

[1] Open up Visual Basic .Net, from the **File** menu select **New** then **Project**.
[2] Right-click on Form1 in the **Project Explorer** window and select Remove Form1 from the pop-up window. (this is an optional step)

[3] You should make a directory for your programs using Windows Explorer and the use the **browse** button to select that directory. Select Console Application and then type in the name of the program, we use Lesson1.

![New Project window]

[3] A project solution appears with a blank module. (Solution's contains projects.)

[4] Now type in the program

```vbc
Module Module1

    Sub Main()

        Dim name As String
        Dim age As Integer
        name = "Tom"
        age = 20
        Console.Write("Hi, my name is ")
        Console.WriteLine(name)
        Console.Write("I am now ")
        Console.Write(age)
        Console.WriteLine(" years old.")
        Console.Read() ' pause

    End Sub

End Module
```

You should now have the following:

```
Option Explicit On

Module Module1

    Sub Main()

        Dim name As String
        Dim age As Integer
        name = "Tom"
        age = 20
        Console.Write("Hi, my name is ")
        Console.WriteLine(name)
        Console.Write("I am now ")
        Console.Write(age)
        Console.WriteLine(" years old.")

    End Sub
```

5) To run the program select start from the Debug menu
The following screen should appear

![Example Program Output]

**Understanding the example program**

Let's go through each line of the example program one by one for understanding. We will put a comment on each line. Comments are used to explain program code. The comments are never executed, they are used just to explain the programming statements. Comments start with a single quotation.

' This is a Comment

Our comments will be in color green to be easily recognized. Comments may start at the beginning of a line or at the end of a programming statement.

\[
\text{age} = 20 \quad \text{' assign the number 20 to the variable age}
\]

Our program explanation is as follows. Programs are made up of programming statements. Programming statements are grouped together in a Sub. Subs are put into Modules. When the VB program starts to run it executes the statements one by one contained in the Sub. When the last statement is executed the program ends. All Subs need a name for identification. The name of our Sub is **Main**. The Main is the first Sub to be executed in a module when the program runs. The round brackets () following Main means Main is a Sub. Round brackets () distinguish a Sub name from a variable name. Names of subs and variables are known as identifiers.
Here is are Main sub with many comments:

```vbnet
Sub Main() ' contains program
    Dim name as String ' declare a string variable to represent a persons name
    Dim age as Integer ' declare an integer variable to represent a persons age
    name = "Tom" ' assign the string "Tom" to the variable name
    age = 20 ' assign the number 20 to the variable age
    Console.Write ("Hi, my name is ") ' print a message to the screen
    Console.WriteLine (name) ' print to the screen the persons name
    Console.Write ("I am now ") ' print a message to the screen
    Console.Write (age) ' print to the screen the persons age
    Console.WriteLine ("years old.") ' print a message to the screen
    Console.Read() ' pause
End Sub ' ends program
```

The program output is the same as before:

```
Hi, my name is Tom
I am now 20 years old.
```
RENAMING PROJECTS AND MODULES

Every time you make a project it gets a default name ConsoleApplication 1. Every time you add a module, it gets a default name like Module 1. You should give projects and modules meaningful names. It’s easy to do. Just go to the project window click on the Module in the Solution Explorer window (on right) and in the Module properties box change the name. You also need to change the Module name in the program.
Changing a project name is trickier and complicated. Just right click on name and select rename then rename it.
If you select properties you will see project name and startup module name. When you change the module name then you must set the startup object to the new name select or just type in.

**Saving your projects and modules**

You need to save all the modules of your project separately from the file menu. From the file menu select **Save Project As** and then **Save Module As**. You only need to use "Save As" once. Next time just use Save Project and save Module.

**VB LESSON1 EXERCISE 1**

Write a program that has 3 variables to display a person's name, address and phone number. Declare and assign values to these variables. Print out to the screen using Console.WriteLine. Call your VB.Net project Project1 and call your module Module1
CONSTANTS

There are situations in a program where you need to assign a value to a variable but you never ever want that variable to change. These variables are called **constants** and are also known as "read only". This means you can read the value again but you can never write a new value to it. You use the keyword **const** to specify that a variable will be a constant and initialize it with a value.

\[
\text{Const Max} = 10
\]

Once you initialize the constant variable with a value it can never change. The constant is only available for use in your module. To let other modules use your constant you proceed the constant declaration with the Public keyword.

\[
\text{Public Const Max} = 10
\]

Once you declare and initialize your constant you use it just like any variable.

\[
\text{Dim x As Integer} \\
x = \text{Max}
\]

Now x has the value 10. Remember you can never assign new values to your constant variable.

\[
x = 20
\]

EXPLICIT OPTION

It is not always necessary to use the **Dim** statement to declare variables. You can use variables right away.

\[
x = 10
\]

It is advised that you declare variables before you use them. To make sure you declare every variable before you use it, you put the

\[
\text{Option Explicit On}
\]

directive at the top of your program. Here's the last program example using **Option Explicit On**
VB LESSON1 EXERCISE 2

Change your Exercise 1 to use **Constants** and the **Explicit Option On**, also assign values to variables when you declare them. Call your module Module2. You can put a second module in your Project1 and in the Project Properties you can set the startup to this module as follows:

```vbnet
Option Explicit On
Module Module1
    Sub Main()
        Dim name As String
        Dim age As Integer
        name = "Tom"
        age = 20
        Console.Write("Hi, my name is ")
        Console.WriteLine(name)
        Console.Write("I am now ")
        Console.Write(age)
        Console.WriteLine(" years old.")
        Console.ReadLine() ' pause
    End Sub
End Module
```
Adding another Module to a Project.

From the Project menu select Add Module.
You need to set startup to this module.

```vbnet
Option Explicit On

Module Module1

    Sub Main()

        Dim name As String
        Dim age As Integer
        name = "Tom"
        age = 20
        Console.WriteLine("Hi, my name is ")
        Console.WriteLine(name)
        Console.WriteLine("I am now ")
        Console.WriteLine(age)
        Console.WriteLine(" years old.")

        Console.Read() ' pause

    End Sub
```

Ready
INPUT BOX

Your program will need to get data values from the keyboard. A good way to get data from the keyboard is to use an Input Box.

These look very professionals and impressive. Each Input Box gets a prompt message to tell the user what to do. Once they enter the data they can click on the OK button or Cancel button.

InputBox is a built in VB function. Built in functions are pre-defined program code that you can use right away. Functions receive values and return values. Using built in functions let you program very fast. You need to know how to use built in functions. Every built in function has a syntax that specifies how to use it. The syntax for the Input Box function is a little overwhelming.

InputBox (prompt[, title] [, default] [, xpos] [, ypos])

Every function gets arguments. Arguments are used to pass values to the function. The InputBox function always needs the prompt value. The other arguments enclosed in square brackets [ ] are optional meaning you do not need to supply values for them. Here are the arguments and their descriptions. Don't be too concerned right now about all the arguments right now.
### Argument Description

**Prompt**
- Required. String expression displayed as the message in the dialog box. The maximum length of prompt is approximately 1024 characters, depending on the width of the characters used. If prompt consists of more than one line, you can separate the lines using a carriage return character (Chr(13)), a linefeed character (Chr(10)), or carriage return–linefeed character combination (Chr(13) & Chr(10)) between each line.

**Title**
- Optional. String expression displayed in the title bar of the dialog box. If you omit title, the application name is placed in the title bar.

**Default**
- Optional. String expression displayed in the text box as the default response if no other input is provided. If you omit default, the text box is displayed empty.

**Xpos**
- Optional. Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If xpos is omitted, the dialog box is horizontally centered.

**Ypos**
- Optional. Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If ypos is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.

It's easy to use the InputBox function, all you have to do is assign the function to a variable. The variable gets the value that the user has entered into the input box.

```
Dim name As String
name = InputBox("What is your name?")
```

Here's a sample program that asks someone for their name and then prints their name to the screen.

```
Sub main()
    Dim name As String
    name = InputBox("What is your name?")
    Debug.Print "Your name is " & name
End Sub
```

Here's the input box

![Input Box]

Here's the program output:

```
Your name is Tom
```
VB LESSON1 EXERCISE 3

Change your exercise 2 to use an Input Box. Ask someone for their name, address and age. Call your module Module3. Add Module 3 to your Project1 and set Module3 as start up.

Message Box’s

A message box lets you display information in a window. The message box is displayed until the OK button is pressed.

To display a message box with a specified text we call the Show method of the MessageBox object.

MessageBox.Show("Your name is " & name)

Here's our example program using a MessageBox

```vbnet
Sub Main()
    Dim name As String
    name = InputBox("What is your name?")
    MsgBox("Your name is " & name)
End Sub
```

Notice we use the '&' operator to join a message and a variable value. Operators are used to do operations with variables. The '&' can also be used to join strings together but not two numeric values.

By default, the message box displays an OK button and a default caption in the title. You may also specify a caption title and what kind of buttons you want.

MessageBox.Show( text, buttons)

MessageBox.Show( text, caption title, buttons)

You may now specify text, title or buttons.
Here are the kinds of buttons you can specify:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbortRetryIgnore</td>
<td>The message box contains Abort, Retry, and Ignore buttons.</td>
</tr>
<tr>
<td>OK</td>
<td>The message box contains an OK button.</td>
</tr>
<tr>
<td>OKCancel</td>
<td>The message box contains OK and Cancel buttons.</td>
</tr>
<tr>
<td>RetryCancel</td>
<td>The message box contains Retry and Cancel buttons.</td>
</tr>
<tr>
<td>YesNo</td>
<td>The message box contains Yes and No buttons.</td>
</tr>
<tr>
<td>YesNoCancel</td>
<td>The message box contains Yes, No, and Cancel buttons.</td>
</tr>
</tbody>
</table>

The MessageBox returns what kind of button that was pressed, into a DialogResult object.

Here are the values returned to the DialogResult object. Each DialogResult has a numeric value.

<table>
<thead>
<tr>
<th>DialogResult</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abort</td>
<td>0</td>
<td>The dialog box return value is Abort (usually sent from a button labeled Abort).</td>
</tr>
<tr>
<td>Cancel</td>
<td>1</td>
<td>The dialog box return value is Cancel (usually sent from a button labeled Cancel).</td>
</tr>
<tr>
<td>Ignore</td>
<td>2</td>
<td>The dialog box return value is Ignore (usually sent from a button labeled Ignore).</td>
</tr>
<tr>
<td>No</td>
<td>3</td>
<td>The dialog box return value is No (usually sent from a button labeled No).</td>
</tr>
<tr>
<td>None</td>
<td>4</td>
<td>Nothing is returned from the dialog box. This means that the modal dialog continues running.</td>
</tr>
<tr>
<td>OK</td>
<td>5</td>
<td>The dialog box return value is OK (usually sent from a button labeled OK).</td>
</tr>
<tr>
<td>Retry</td>
<td>6</td>
<td>The dialog box return value is Retry (usually sent from a button labeled Retry).</td>
</tr>
<tr>
<td>Yes</td>
<td>7</td>
<td>The dialog box return value is Yes (usually sent from a button labeled Yes).</td>
</tr>
</tbody>
</table>

Here is an example using **YesNo** Buttons and return’s value assigned to a **MsgBoxResult** variable.

```vbnet
Sub Main()
    Dim name As String
    name = InputBox("What is your name?")
    Dim result As MsgBoxResult
    result = MsgBox("Your name is " & name & "?", MessageBoxButtons.YesNo)
    Console.WriteLine(result)
End Sub
```
The output is 6 as expected when the Yes button pressed.

![Image of a dialog box with the message "Your name is Tom"

Line String Constants

The following string constants are used to specify a new line, tab etc. The string constants represent a `Chr` function that is passed a numeric value to specify a new line, tab etc.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VbCr</td>
<td><code>Chr(13)</code></td>
<td>Carriage return</td>
</tr>
<tr>
<td>VbCrLf</td>
<td><code>Chr(13)</code> &amp; <code>Chr(10)</code></td>
<td>Carriage return–linefeed combination</td>
</tr>
<tr>
<td>VbFormFeed</td>
<td><code>Chr(12)</code></td>
<td>Form feed; not useful in Microsoft Windows</td>
</tr>
<tr>
<td>VbLf</td>
<td><code>Chr(10)</code></td>
<td>Line feed</td>
</tr>
<tr>
<td>VbNewLine</td>
<td><code>Chr(13)</code> &amp; <code>Chr(10)</code> or <code>Chr(10)</code></td>
<td>Platform-specific newline character; whatever is appropriate for the platform</td>
</tr>
<tr>
<td>VbNullChar</td>
<td><code>Chr(0)</code></td>
<td>Character having the value 0</td>
</tr>
<tr>
<td>VbNullString</td>
<td>String having value 0</td>
<td>Not the same as a zero-length string (&quot;&quot;); used for calling external procedures</td>
</tr>
<tr>
<td>VbTab</td>
<td><code>Chr(9)</code></td>
<td>Horizontal tab</td>
</tr>
<tr>
<td>VbVerticalTab</td>
<td><code>Chr(11)</code></td>
<td>Vertical tab; not useful in Microsoft Windows</td>
</tr>
</tbody>
</table>

**VB LESSON1 EXERCISE 4**

Change your exercise 1 to use include a `MessageBox`. Ask the user to enter their name, address and age using the Input Box and then display them in an `MsgBox`. Use a line feed character `Chr(10)` or `VbCrLf` to separate lines in the `MsgBox`. Call your module Module4
IMPORTANT

You should use all the material in all the lessons to do the questions and exercises. If you do not know how to do something or have to use additional books or references to do the questions or exercises, please let us know immediately. We want to have all the required information in our lessons. By letting us know we can add the required information to the lessons. The lessons are updated on a daily bases. We call our lessons the "living lessons". Please let us keep our lessons alive.

E-Mail all typos, unclear test, and additional information required to:

courses@cstutoring.com

E-Mail all attached files of your completed exercises to:

students@cstutoring.com

This lesson is copyright (C) 2006 by The Computer Science Tutoring Center "www.cstutoring.com"
This document is not to be copied or reproduced in any form. For use of student only