



Programming With
Microsoft Visual
Basic 2015

Variables and Math Operations

Assistant Lecturer

Nawfal Turki Obais

3^{Lec}

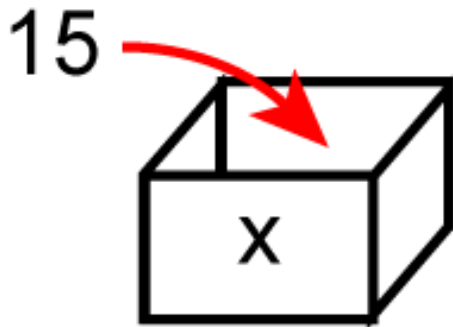
Variables

A **variable** is a location in memory, which has been specifically reserved for holding data.i.e. A variable is a '**data box**'. A variable thus binds the stored data to a name. Which enables **convenient access** at a later time. Variables can be assigned values, via 'assignment': Such an assignment **operation** takes the form: *variable_name* = *value* Here, '=' is the **assignment** operator (NOT the 'equality' operator) Examples:

Variables *Example 1:*

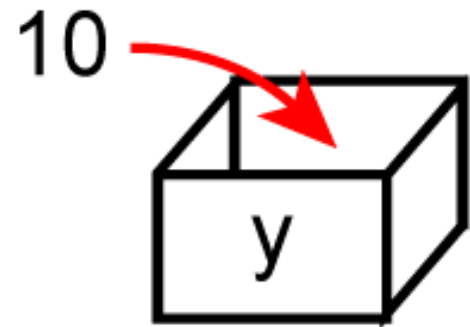
'Assign 15 to x

$x = 15$



'Assign 10 to y

$y = 10$



Variables Declaration and assignment

Prior to use, in VB variables must first be declared...This reserves (creates) a place in your computer's memory. The VB syntax for a local variable declaration:

Dim variable_name **As** data_type

Dim is a VB keyword for declaring a variable; variable_name is the name of the variable; **As** is a VB keyword for specifying the data_type... the type of data the variable will hold. Example of variable use:

Adding 10 + 15...

Variables Declaration and assignment *Example 2:*

'Declare integer variables x, y, and z

Dim x, y, z **As Integer**

'Assign 15 to x

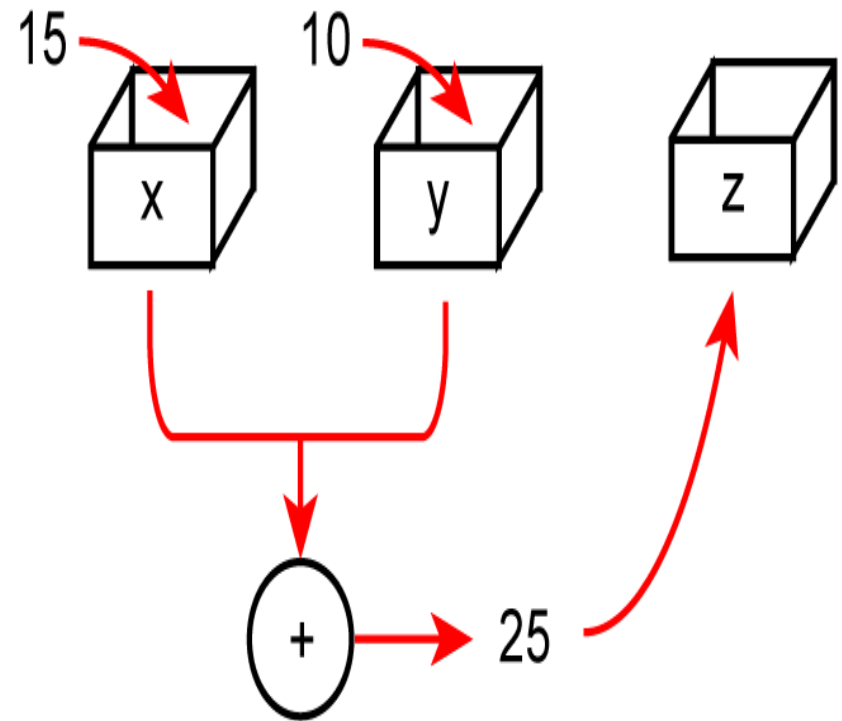
$x = 15$

'Assign 10 to y

$y = 10$

'Add x to y; assign the result to z

$z = x + y$



Variables Declaration and assignment

For a constant, both memory location and the stored value are fixed. The VB syntax for declaring a constant is:

Const const_name **As** Data_Type = value

Const is a keyword declaring the constant; const_name and value are the name and value of the constant; Data_Type is the type of constant.

Note: variables can also be given initial values...In both cases, this is called ‘initialization’.

Example 4 : Const x As Integer = 10

Example 5: Const PI As Double = 3.14

Rules for naming variables in Visual Basic

1. **Unique** within scope
2. variable \leq **32** characters (other languages may have different length restrictions)
3. Begin with a **letter**
4. No embedded **spaces** or many **special characters** (. , “ - \$ # * and others). The underscore _ is a valid character.
5. Cannot be a reserved word like **messagebox**
6. Use **upper and lower** case with purpose. Once a variable is declared you do not have to be concerned with upper/lower case. The editor recognizes words that are the same except for case and makes them all the same for you. (*use of upper and lower case differ between languages*)
7. It is always good programming practice to use names that are **descriptive or mnemonic**.

Rules for naming variables in Visual Basic

Exercise1: Fill the blanks with suitable answer

Variable Name	Valid?	Invalid?	Good Variables Name? If not why?
GPA Count#1 \$grosspay			
GradePointAverage			
Attempted.Hours			
Attempted-Hours			
Interest Rate			
A2			
2A			
End			
Printitout			
MONEYAFTERALLTAXES			

Data Types

Data Type	Name	Memory	Range
Byte	Byte	1 byte	0 ~ 255
Short	Short Integer	2 bytes	-32,768 ~ 32,768
Integer	Integer	4 bytes	-2,147,483,648 ~ 2,147,483,648
Long	Long Integer	8 bytes	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,808
Single	Single Precision Floating point	4 bytes	-3.4028235 E38 ~ 3.4028235 E38
Double	Double-precision Floating point	8 bytes	-1.7976931348623157 E308 ~ 1.7976931348623157 E308
Decimal	Fixed point decimal	16 bytes	-7.9228162514264337593543950335 E28 ~ 7.9228162514264337593543950335 E28
Char	Character	2 bytes	1 char (0 ~ 65535, Unicode)
String	Char string	2 bytes/char	Max, 2 billion characters
Date	Date	8 bytes	AD Jan 1, 0001 - AD Dec 31, 9999
Boolean	Boolean	2 byte	True or False
Object	Object	4 byte	Arbitrary data type reference

Data Type Example

```
Public Class Form1
```

```
    Dim b As Byte
```

```
    Dim n As Integer
```

```
    Dim si As Single
```

```
    Dim d As Double
```

```
    Dim da As Date
```

```
    Dim c As Char
```

```
    Dim s As String
```

```
    Dim bl As Boolean
```

```
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
    End Sub
```

```
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
```

```
    End Sub
```

```
End Class
```

The Scope and Lifetime of a Variable

Besides a name, a data type, and an initial value, every variable also has a *scope* and a *lifetime*. A variable's *scope* indicates where the variable can be used in an application's code, and its *lifetime* indicates how long the variable remains in the computer's internal memory. Variables can have *class scope*, *procedure scope*, or block scope. However, most of the variables used in an application will have procedure scope. This is because fewer unintentional errors occur in applications when the variables are declared using the minimum scope needed, which usually is procedure scope.

The Scope and Lifetime of a Variable

1-Variables with Procedure Scope:

When you declare a variable in a procedure, the variable is called a ***procedure-level variable***. Procedure-level variables have ***procedure scope*** because they can be used only by the procedure in which they are declared. Procedure-level variables are typically declared at the beginning of a procedure, and they remain in the computer's internal memory only while the procedure is running. Procedure-level variables are ***removed*** from memory when the procedure in which they are declared ***ends***. In other words, a procedure-level variable has the ***same lifetime*** as the procedure that declares it. As mentioned earlier, most of the variables in your applications will be procedure-level variables.

The Scope and Lifetime of a Variable

```
Private Sub btnRate8_Click(sender As Object, e As EventArgs)  
Handles btnRate8.Click
```

```
    ' calculates and displays an 8% commission
```

```
    ' the Dim statements declare two procedure-level  
    ' variables that can be used only within the  
    ' btnRate8_Click procedure
```

```
    Dim dblSales As Double
```

```
    Dim dblComm8 As Double
```

removed from memory
when the btnRate8_Click
procedure ends

```
End Sub
```

```
Private Sub btnRate10_Click(sender As Object, e As EventArgs)  
Handles btnRate10.Click
```

```
    ' calculates and displays a 10% commission
```

```
    ' the Dim statements declare two procedure-level  
    ' variables that can be used only within the  
    ' btnRate10_Click procedure
```

```
    Dim dblSales As Double
```

```
    Dim dblComm10 As Double
```

removed from memory
when the btnRate10_Click
procedure ends

```
End Sub
```


The Scope and Lifetime of a Variable

2-Variables with Class Scope

In addition to declaring a variable in a procedure, you can also declare a variable in the *form's Declarations section*, which *begins* with the *Public Class* clause and *ends* with the *End Class* clause. When you declare a variable in the form's Declarations section, the variable is called a *class-level variable* and it has *class scope*. *Class-level variables can be used by all of the procedures in the form*, including the procedures associated with the controls contained on the form. Class-level variables retain their values and remain in the computer's internal memory until the *application ends*. In other words, a class-level variable has the *same lifetime* as the *application itself*.

The Scope and Lifetime of a Variable

```
Public Class frmMain
    ' class-level variable for accumulating the scores
    Private decTotal As Decimal

    Private Sub btnAdd_Click(sender As Object,
        e As EventArgs) Handles btnAdd.Click
        ' totals the scores entered by the user

        ' procedure-level variable for getting each score
        Dim decScore As Decimal

        ' total the scores and display the result
        Decimal.TryParse(txtScore.Text, decScore)
        decTotal = decTotal + decScore
        lblTotal.Text = Convert.ToString(decTotal)

        txtScore.Focus()
    End Sub
```

class-level
variable
declared in
the form's
Declarations
section

procedure-
level variable
declared in the
btnAdd_Click
procedure

Data Type Conversion

1-Implicit Type Conversion

As you learned earlier, the data type of the value assigned to a memory location should be the same as the data type of the memory location itself. If the value's data type does not match the memory location's data type, the computer uses a process called *implicit type conversion* to convert the value to fit the memory location.

For example, when processing the statement

Dim dblLength As **Double** = 9, the computer converts the integer 9 to the Double number 9.0 before storing the value in the dblLength variable. *When a value is converted from one data type to another data type that can store either larger numbers or numbers with greater precision, the value is said to be promoted. In this case, if the dblLength variable is used subsequently in a calculation, the results of the calculation will not be adversely affected by the implicit promotion of the number 9 to the number 9.0.*

Data Type Conversion

2-Explicit Type Conversion

Type Conversion Rules

1. *Strings will not be implicitly converted to numbers.* The Code Editor will display a warning message when a statement attempts to use a string where a number is expected.

Incorrect: `dblRadius = txtRadius.Text`

Correct: `dblRadius = Convert.ToDouble(txtRadius.Text)`

2. *Numbers will not be implicitly converted to strings.* The Code Editor will display a warning message when a statement attempts to use a number where a string is expected.

Incorrect: `lblArea.Text = dblArea`

Correct: `lblArea.Text = Convert.ToString(dblArea)`

Correct: `lblArea.Text = Format(dblArea, "standard")`

Data Type Conversion

3. *Wider data types will not be implicitly demoted to narrower data types.* The Code Editor will display a warning message when a statement attempts to use a wider data type where a narrower data type is expected.

Incorrect: `Dim decRate As Decimal = 0.05`

Correct: `Dim decRate As Decimal = 0.05D`

Correct: `Dim decRate As Decimal = Convert.ToDecimal(0.05)`

4. *Narrower data types will be implicitly promoted to wider data types.*

Correct: `dblAverage = dblTotal / intNum`

Arithmetic (Math)operation

Operator	Meaning	Example
+	addition	$6 + 5 = 11$
-	subtraction	$6 - 5 = 1$
*	multiplication	$6 \times 5 = 30$
/	division	$6 / 5 = 1.2$
\ (or ¥)	quotient (int. division)	$6 \setminus 5 = 1$
Mod	remainder	$6 \text{ Mod } 5 = 1$
^	exponential	$6^5 = 7776$

Math statement

Question: what does the statement, ' $x = x + 1$ ' do? Thinking in terms of algebra, this is a nonsense statement. Since '=' is defined equality... But x is never equal to $x + 1$! However, if we as instead think in VB, it makes perfect sense! Remember...here, '=' is the assignment operator. Thus, ' $x = x + 1$ ' tells the computer to: First, get the value stored in variable x . Then, add 1 to this value.

Math statement

Lastly, store the result in variable **x**. For example, assume **x** starts out as **10**:

```
Dim x As Integer = 10
```

```
x = x + 1
```

During run-time, the right side is first evaluated to yield **11**. Then, this result (**11**) is passed to the left side (**x**).

So, the overall result is to set:

x = 11.

Math statement

More generally, a math statement takes the form:

left_side = right_side

Where, 'left_side' is a variable...

While 'right_side' is a mathematical expression. At run-time, the right_side is evaluated...And then passed to the left_side. For instance, as a result of the statement:

$z = 2 * 3$

First, the right side is evaluated (yielding 6).

Then, the result is passed to z (setting z equal to 6).

Math statement

What about a compound statement (several math ops):

$$x = 3 * 2 + 1 ?$$

If we perform the multiplication first, we get :

$$x = 6 + 1 = 7.$$

If we add first, we get :

$$x = 3 * 3 = 9.$$

Which is correct?

Operator Priority

In VB, the order of evaluation of math operators (ops) is determined by **precedence**.

For arithmetic, the order of evaluation is (first to last):

1. Exponentiation (^)
2. Unary identity and negation (+, -) Such as the '-' in 'x = -6'
3. Multiplication and floating-point division (*, /)
4. Integer division (\)
5. Modulus arithmetic (Mod)
6. Addition and subtraction (+, -)

Operator Priority

So, for our earlier example:

$$x = 3 * 2 + 1$$

$$\rightarrow 6 + 1$$

$$\rightarrow 7.$$

What happens when the compiler encounters several successive ops with equal precedence?

The compiler evaluates the ops in order, from left to right.

$$\text{Ex: } x = 6 * 2 / 4 * 3 \rightarrow 12 / 4 * 3 \rightarrow 3 * 3 \rightarrow 9$$

Operator Priority

What if we want to do the addition first...?

- VB's default operation order can be over-ridden easily!
 - By simply adding parentheses.
 - In particular, operations enclosed by parenthesis are evaluated first...

Examples:

Our example, stated as: $z = 3 * (2 + 1) = 3 * 3 = 9$

However, stated as: $y = (3 * 2) + 1 = 6 + 1 = 7$

Thus, parenthesis provide simple program control, during execution.

Operator Priority

This also applies to nested parentheses...

Parentheses *inside* of parentheses (expressions with several 'layers')

The most 'internal' operations are performed first.

Example : $X = (((2 + 1) * 3) + ((7 + 6) - 4)) * 5$ ← inner layer

→ $= ((3 * 3) + (13 - 4)) * 5$ ← middle layer

→ $= (9 + 9) * 5$ ← outer layer

→ $= 18 * 5$

→ $= 90.$

Work in Lab

- Make simple calculator and apply variables constant
- Make project to apply and show the scope of variables

Step 1. Place Controls

Place 5 Labels, 2 TextBoxes, and 8 Buttons

The screenshot shows a Windows Form titled 'Form1'. It contains the following controls:

- Label1, Label2, Label3, Label4, and Label5 are positioned at the top.
- Two text boxes are located below the labels.
- Eight buttons, labeled Button1 through Button8, are arranged in two rows at the bottom.

Step 2. Set Properties

Note: Use 'standard' naming for important Controls:
→ Buttons: 'btnOperation' (Ex: btnIntDiv)
→ TextBoxes: 'txtUse' (Ex: txtA, txtB)
→ Label: lblOutput

The screenshot shows a 'Simple Calculator' application window. It features the following elements:

- A header section titled 'Input Values for A and B'.
- Input fields for 'A =' and 'B ='.
- A 'Result' field displaying a question mark '?'.
- A grid of buttons for arithmetic operations: '+', '-', '*', '/', '\', 'mod', '^', and an 'Exit' button.

Work in Lab

Write and discuss with your friend and TAS (teaching assistant in LAB) a project to find the solution of Z where Z equal to

$$R=8$$

$$Z=3*5-8+3\backslash 1/1\text{mod } 2 +R$$



Thank You