Chapter 3 – Variables, Input, and Output

3.1 Numbers
3.2 Strings
3.3 Input and Output
3.1 Numbers

- Arithmetic Operations
- Variables
- Incrementing the Value of a Variable
- Mathematical Functions
- The Integer Data Type
Numbers (continued)

- Multiple Declarations
- Two Integer-Valued Operators
- Parentheses
- Three Types of Errors
- The Error List Window
Arithmetic Operations

- Numbers are called **numeric literals**
- Five arithmetic operations in Visual Basic
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - ^ exponentiation
Numeric Expressions

2 + 3
3 * (4 + 5)
2 ^ 3
Displaying Numbers

Let $n$ be a number or a numeric expression.

The statement

```c
lstBox.Items.Add(n)
```

displays the value of $n$ in the list box.
Example 1: Form
Example 1: Code and Output

Private Sub btnCompute_Click(...) Handles btnCompute.Click
    lstResults.Items.Add(5)
    lstResults.Items.Add(2 * 3)
    lstResults.Items.Add((2 ^ 3) - 1)
End Sub

Output

5 in list
6 box
7
A **numeric variable** is a name to which a number can be assigned.

Examples:

- speed
- distance
- interestRate
- balance
Variables

• Declaration:

\[ \text{Dim speed As Double} \]

• Assignment:

\[ \text{speed} = 50 \]
Initialization

- Numeric variables are automatically initialized to 0:
  
  ```
  Dim varName As Double
  ```

- To specify a nonzero initial value
  
  ```
  Dim varName As Double = 50
  ```
Numeric Expressions

Numeric variables can be used in numeric expressions.

```vbs
Dim balance As Double = 1000
lstBox.Items.Add(1.05 * balance)
```

Output: 1050
Assignment Statement

Dim numVar1 As Double = 5
Dim numVar2 As Double = 4
numVar1 = 3 * numVar2
lstBox.Items.Add(numVar1)

Output: 12
Incrementing

- To add 1 to the numeric variable \( \text{var} \)
  \[
  \text{var} = \text{var} + 1
  \]
- Or as a shortcut
  \[
  \text{var} += 1
  \]
- Or as a generalization
  \[
  \text{var} += \text{numeric expression}
  \]
Built-in Functions

Functions *return a value*

Math.Sqrt(9) returns 3

Int(9.7) returns 9

Math.Round(2.7) returns 3
Integer Data Type

- Variables of type Double can be assigned both whole numbers and numbers with decimals.
- The statement

  ```vbnet
  Dim varName As Integer
  ```

declares a numeric variable that can only be assigned whole number values between about -2 billion and 2 billion.
Multiple Declarations

**Dim a, b As Double**

Two other types of multiple-declaration statements are

**Dim a As Double, b As Integer**
**Dim c As Double = 2, b As Integer = 5**
Two Integer-Valued Operators

- Integer division (denoted by \) is similar to ordinary long division except that the remainder is discarded.
- The Mod operator returns only the integer remainder after long division.

\[
23 \div 7 = 3 \quad 23 \text{ Mod } 7 = 2 \\
8 \div 2 = 4 \quad 8 \text{ Mod } 2 = 0
\]
Parentheses

- Parentheses should be used liberally in numeric expressions.
- In the absence of parentheses, the operations are carried out in the following order: ^, *, /, \, Mod, + and -.
Three Types of Errors

- Syntax error
- Runtime error
- Logic error
Some Types of Syntax Errors

• Misspellings
  lstBox.Items.Add(3)

• Omissions
  lstBox.Items.Add(2 + )

• Incorrect punctuation
  Dim m; n As Integer
Overflow error

Dim numVar As Integer = 1000000
numVar = numVar * numVar
A Logical Error

Dim average As Double
Dim m As Double = 5
Dim n As Double = 10
average = m + n / 2

Value of average will be 10. Should be 7.5.
Error List Window

\textbf{Dim} \textit{m; n As Double}

\textit{lstResults.Items.Add(5}
\textit{lstResults.Items.Add(a)
3.2 Strings

- Variables and Strings
- Using Text Boxes for Input and Output
- Option Explicit and Option Strict
- Concatenation
- String Properties and Methods
- The Empty String
Strings (continued)

• Initial Value of a String
• Widening and Narrowing
• Internal Documentation
• Line Continuation
• Scope of a Variable
• Auto Correction
String Literal

A **string literal** is a sequence of characters surrounded by quotation marks.

Examples:

"hello"
"123-45-6789"
"#ab cde?"
String Variable

A **string variable** is a name to which a string value can be assigned.

Examples:

- country
- ssn
- word
- firstName
String Variable (continued)

- **Declaration:**
  
  ```
  Dim firstName As String
  ```

  - variable name
  - data type

- **Assignment:**
  
  ```
  firstName = "Fred"
  ```
You can declare a string variable and assign it a value at the same time.

Dim firstName As String = "Fred"
Add Method

Let $str$ be a string literal or variable. Then,

```
1stBox.Items.Add(str)
```

displays the value of $str$ in the list box.
String Variable

You can assign the value of one string variable to another.

```vba
Dim strVar1 As String = "Hello"
Dim strVar2 As String = "Goodbye"
strVar2 = strVar1
lstOutput.Items.Add(strVar2)
```

Output: Hello
Variables and Strings

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim president As String
    president = "George Washington"
    lstOutput.Items.Add("president")
    lstOutput.Items.Add(president)
End Sub

Output:  president
         George Washington
Using Text Boxes for Input and Output

- The contents of a text box is always a string.

- Input example:
  
  ```csharp
  strVar = txtBox.Text
  ```

- Output example:
  
  ```csharp
  txtBox.Text = strVar
  ```
Data Conversion

Because the contents of a text box is always a string, sometimes you must convert the input or output.

```vbnet
dblVar = CDbl(txtBox.Text)
```
converts a String to a Double

```vbnet
txtBox.Text = CStr(numVar)
```
converts a number to a string
Option Strict

• Visual Basic allows numeric variables to be assigned strings and vice versa, a poor programming practice.

• To prevent such assignments, set Option Strict to On in the Options dialog box.
Option Strict (continued)

- Select *Options* from the *Tools* menu
- In left pane, expand Projects and Solution
- Select VB Defaults
- Set Option Strict to On
Option Strict (continued)
With Option Strict On

Dim dblVar As Double, intVar As Integer
Dim strVar As String

Not Valid:
intVar = dblVar
dblVar = strVar
strVar = intVar

Replace with:
intVar = CInt(dblVar)
dblVar = CDbl(strVar)
strVar = CStr(intVar)
Concatenation

Combining two strings to make a new string

quote1 = "We'll always "
quote2 = "have Paris."
quote = quote1 & quote2
txtOutput.Text = quote & " - Humphrey Bogart"

Output:
We'll always have Paris. - Humphrey Bogart
Appending

• To append \textit{str} to the string variable \textit{var}
  \begin{verbatim}
  var = var & str
  \end{verbatim}

• Or as a shortcut
  \begin{verbatim}
  var &= str
  \end{verbatim}
Appending Example

Dim var As String = "Good"
var &= "bye"
txtBox.Text = var

Output: Goodbye
Comment on Example 4

Consider

txtOutput.Text = numOfKeys & " keys"

The ampersand automatically converts
numOfKeys into a string before concatenating.
We do not have to convert numOfKeys with CStr.
String Properties and Methods

"Visual".Length is 6.

"Visual".ToUpper is VISUAL.

"123 Hike".Length is 8.

"123 Hike".ToLower is 123 hike.

"a" & " bcd ".Trim & "efg" is abcdefg.
Positions in a String

Positions of characters in a string are numbered 0, 1, 2, ....

Consider the string “Visual Basic”.
Position 0: V
Position 1: i
Position 7: B
Substring “al” begins at position 4
Let $str$ be a string.

$\text{str}.\text{Substring}(m, n)$ is the substring of length $n$, beginning at position $m$ in $str$.

“Visual Basic”.Substring(2, 3) is “sua”

“Visual Basic”.Substring(0, 1) is “V”
IndexOf Method

Let $str1$ and $str2$ be strings.

$str1$.IndexOf($str2$)
is the position of the first occurrence of $str2$ in $str1$.
( ciclo: Has value -1 if $str2$ is not a substring of $str1$.)

"Visual Basic".IndexOf("is") is 1.
"Visual Basic".IndexOf("si") is 9.
"Visual Basic".IndexOf("ab") is -1.
The Empty String

- The string "", which has no characters, is called the **empty string** or the **zero-length string**.
- The statement `lstBox.Items.Add("")` skips a line in the list box.
- The contents of a text box can be cleared with either the statement
  
  ```csharp
  txtBox.Clear()
  ```

  or the statement
  
  ```csharp
  txtBox.Text = ""
  ```
Initial Value of a String Variable

• By default the initial value is the keyword Nothing.
• Strings can be given a different initial value as follows:

```vbnet
Dim name As String = "Fred"
```
Widening

- Widening: assigning an Integer value to a Double variable
- Widening always works. (Every Integer value is a Double value.)
- No conversion function needed.
Narrowing

- Narrowing: assigning a Double value to an Integer variable
- Narrowing might not work. (Not every Double value is an Integer value.)
- Narrowing requires the Cint function.
Private Sub btnCompute_Click ("")
    Handles btnCompute.Click
    'Calculate the balance in an account
    Dim rate As Double  'Annual rate of interest
    Dim curBalance As Double  'Current balance
1. Other people can easily understand the program.
2. You can understand the program when you read it later.
3. Long programs are easier to read because the purposes of individual pieces can be determined at a glance.
Line Continuation

A long line of code can be continued on another line by using an underscore (\_) preceded by a space

```python
msg = "I'm going to make " & _
    "him an offer he can't refuse."```

Implicit Line Continuation

The line continuation character can be omitted after a comma, ampersand, or arithmetic operator.

```python
msg = "I'm going to make " & 
    "him an offer he can't refuse."

average = sumOfNumbers / 
    numberOfNumbers
```
Scope

• The **scope** of a variable is the portion of the program that can refer to it.

• Variables declared inside an event procedure are said to have **local scope** and are only available to the event procedure in which they are declared.
Scope (continued)

• Variables declared outside an event procedure are said to have class-level scope and are available to every event procedure.

• Usually declared after

  `Public Class formName`

  (In Declarations section of Code Editor.)
Auto Correction

Private Sub Button1_Click(sender As Object, e As EventArgs)
    txtBox.Text = 1234
End Sub

Option Strict On disallows implicit conversions from 'Integer' to 'String'.

Replace '1234' with 'CStr(1234)'.

Private Sub Button1_Click(sender As Object, e As EventArgs)
    txtBox.Text = CStr(1234)
End Sub
Automatic Colorization

Comments – green
String literals – maroon
Keywords – blue
Class Name – turquoise

Note: Examples of keywords are Handles, Sub, and End. Examples of class names are Form1, Math, and MessageBox.
3.3 Input and Output

- Formatting Numeric Output
- Dates as Input and Output
- Using a Masked Text Box for Input
- Getting Input from an Input Dialog Box
- Using a Message Dialog Box for Output
- Named Constants
## Formatting Numeric Output with the ToString Method

<table>
<thead>
<tr>
<th>Method</th>
<th>String Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(12345.628).ToString(&quot;N1&quot;)</code></td>
<td>12,345.6</td>
</tr>
<tr>
<td><code>(12345.628).ToString(&quot;C2&quot;)</code></td>
<td>$12,345.63</td>
</tr>
<tr>
<td><code>(0.183).ToString(&quot;P0&quot;)</code></td>
<td>18%</td>
</tr>
</tbody>
</table>
Dates as Input and Output

- Date literal: \#7/4/1776\#
- Declarations:

\[
\begin{align*}
\text{Dim indDay As Date} \\
\text{Dim d As Date = CDate(txtBox.Text)} \\
\text{Dim indDay As Date = #7/4/1776}\#
\end{align*}
\]
Masked Text Box Control

Similar to an ordinary text box, but has a Mask property that restricts what can be typed into the masked text box.

Tasks button
Masked Text Box Control

Click on the Tasks button to reveal the Set Mask property.

Click Set Mask to invoke the Input Mask dialog box.
Input Mask Dialog Box

Select a predefined mask description from the list below or select Custom to define a custom mask.

<table>
<thead>
<tr>
<th>Mask Description</th>
<th>Data Format</th>
<th>Validating Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric (5-digits)</td>
<td>12345</td>
<td>Int32</td>
</tr>
<tr>
<td>Phone number</td>
<td>(574) 555-0123</td>
<td>(none)</td>
</tr>
<tr>
<td>Phone number no area code</td>
<td>555-0123</td>
<td>(none)</td>
</tr>
<tr>
<td>Short date</td>
<td>12/11/2003</td>
<td>DateTime</td>
</tr>
<tr>
<td>Short date and time (US)</td>
<td>12/11/2003 11:20</td>
<td>DateTime</td>
</tr>
<tr>
<td>Social security number</td>
<td>000-00-1234</td>
<td>(none)</td>
</tr>
<tr>
<td>Time (European/Military)</td>
<td>23:20</td>
<td>DateTime</td>
</tr>
<tr>
<td>Time (US)</td>
<td>11:20</td>
<td>DateTime</td>
</tr>
<tr>
<td>Zip Code</td>
<td>98052-6399</td>
<td>(none)</td>
</tr>
<tr>
<td>&lt;Custom&gt;</td>
<td></td>
<td>(none)</td>
</tr>
</tbody>
</table>

Mask: [Input field]

Preview: [Input field]

Use ValidatingType: [Check box]

OK | Cancel
A Mask setting is a sequence of characters, with 0, L, and & having special meanings.

- 0  Placeholder for a digit.
- L  Placeholder for a letter.
- &  Placeholder for a character
Sample Masks

- State abbreviation: LL
- Phone number: 000-0000
- Social Security Number: 000-00-0000
- License plate: &&&&&&&
Getting Input from an Input Dialog Box

```plaintext
stringVar = InputBox(prompt, title)
fullName = InputBox("Enter your full name.", "Name")
```
Using a Message Dialog Box for Output

```csharp
MessageBox.Show(prompt, title)
MessageBox.Show("Nice try, but no cigar.", "Consolation")
```
Named Constants

- Declared with
  \[\text{Const } \textit{CONSTANT\_NAME} \text{ As } \textit{DataType} = \text{value}\]
- Value cannot be changed.

Examples:

\[
\begin{align*}
\text{Const } \textit{MIN\_VOTING\_AGE} \text{ As Integer} & = 18 \\
\text{Const } \textit{INTEREST\_RATE} \text{ As Double} & = 0.035 \\
\text{Const } \textit{TITLE} \text{ As String} & = "\text{Visual Basic}\"
\end{align*}
\]