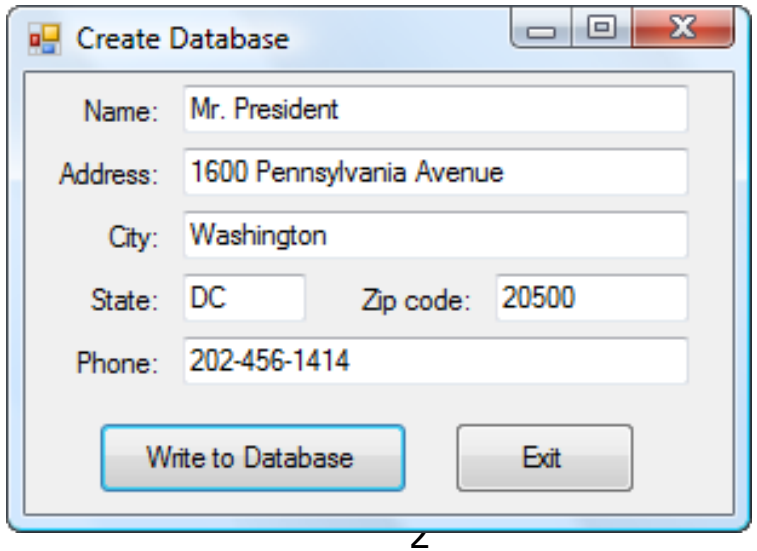
Review of Chapter 2

How to Develop a VB Application

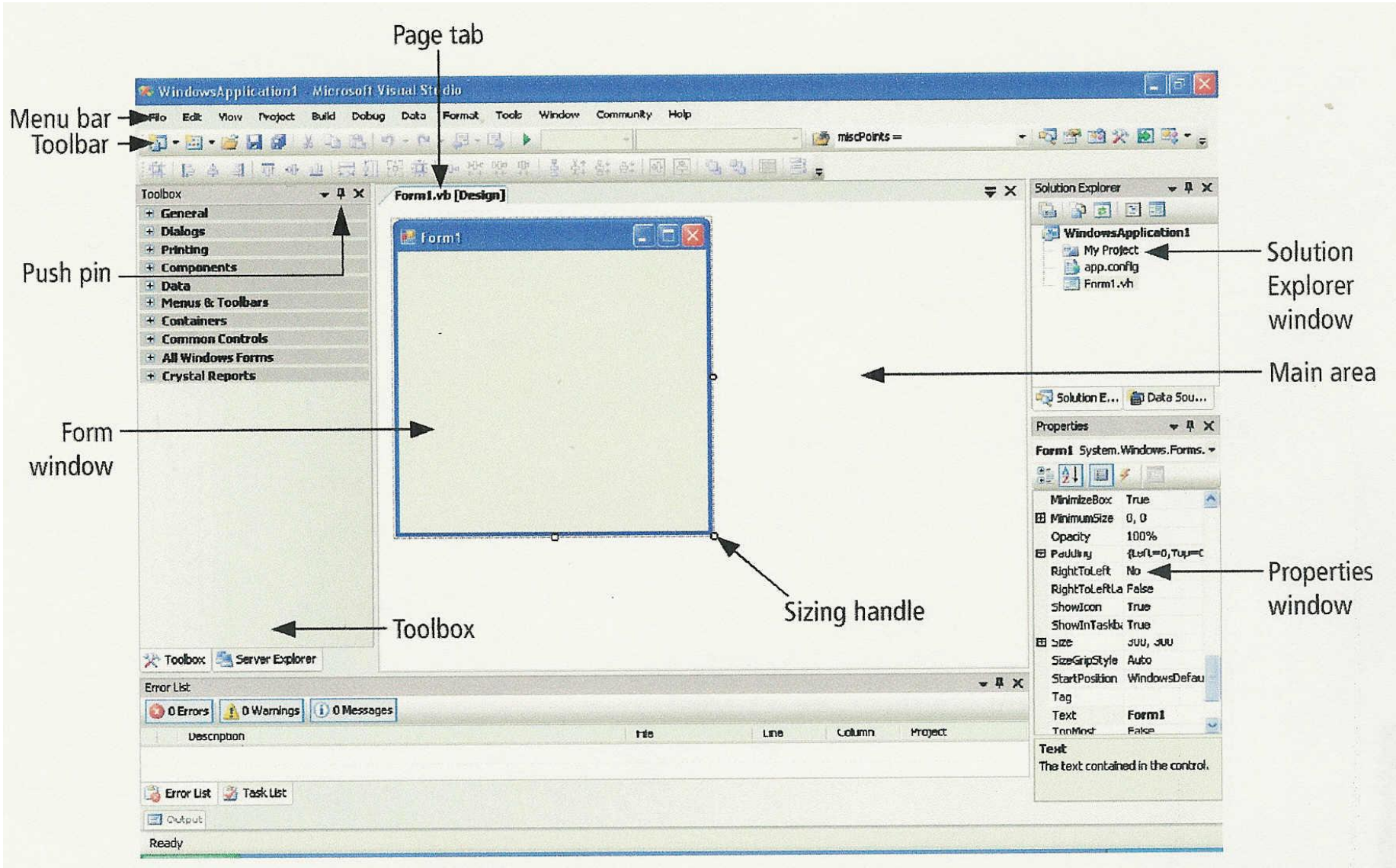
- Design the Interface for the user
  - Literally draw the GUI
  - Drag buttons/text boxes/etc onto form
- Determine which events the controls on the window should recognize
- Write the code for those events



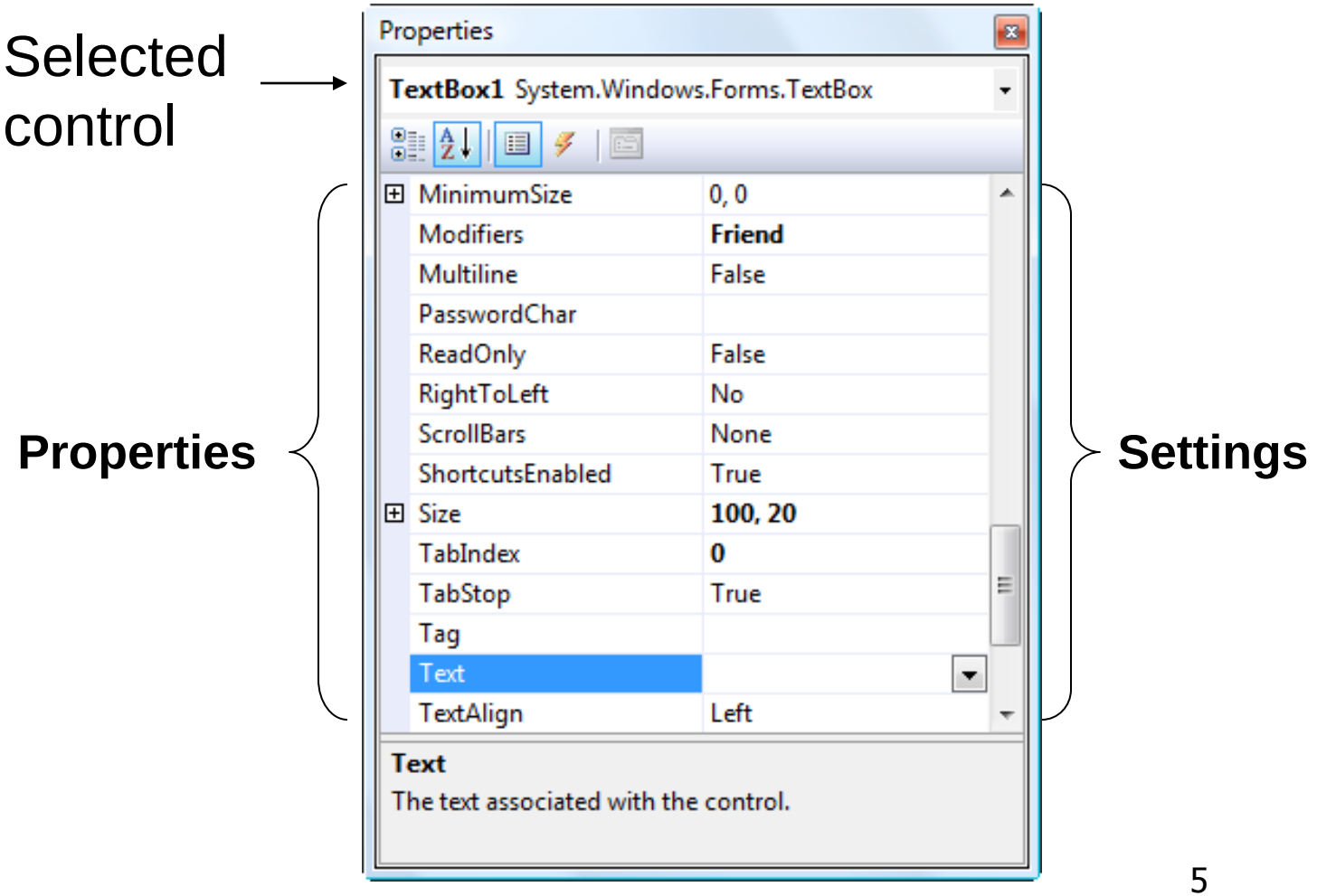
What happens when program is running

1. VB monitors the controls for events
2. If event occurs, it runs procedures assigned to that event
3. If no event exists, it goes back to #1.

Initial Visual Basic Screen



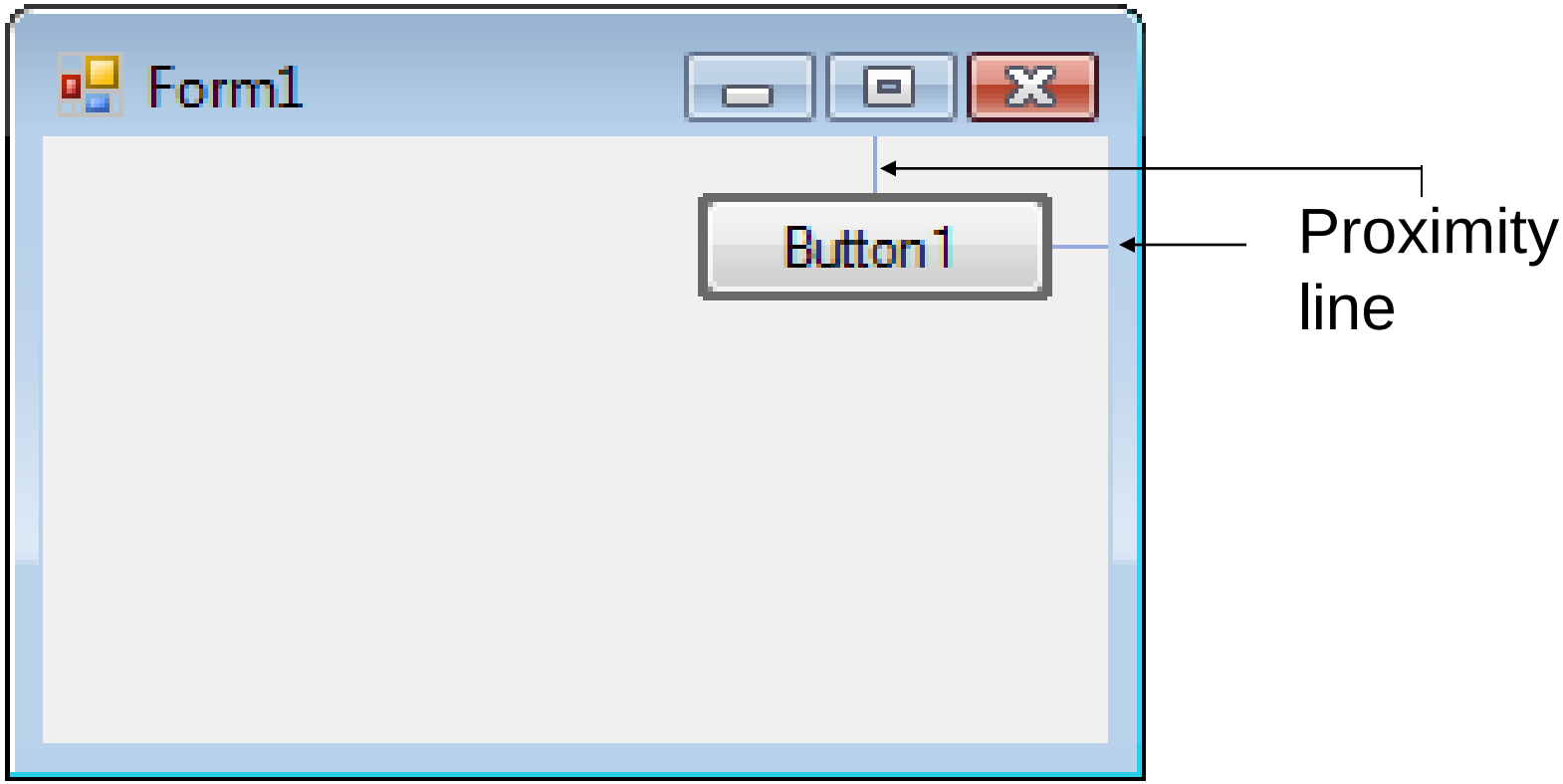
Properties Window



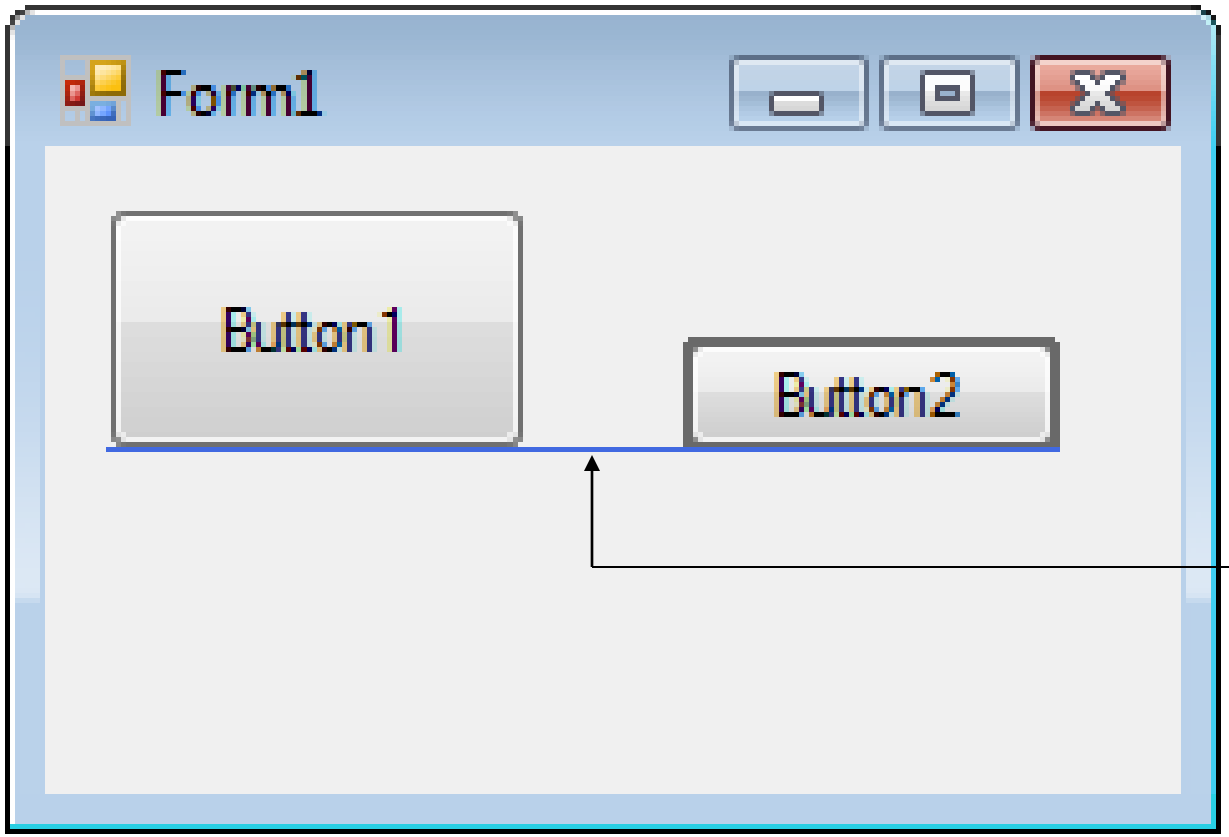
Control Name Prefixes

Control	Prefix	Example
button	btn	btnCompute
label	lbl	lblAddress
text box	txt	txtAddress
list box	lst	lstOutput

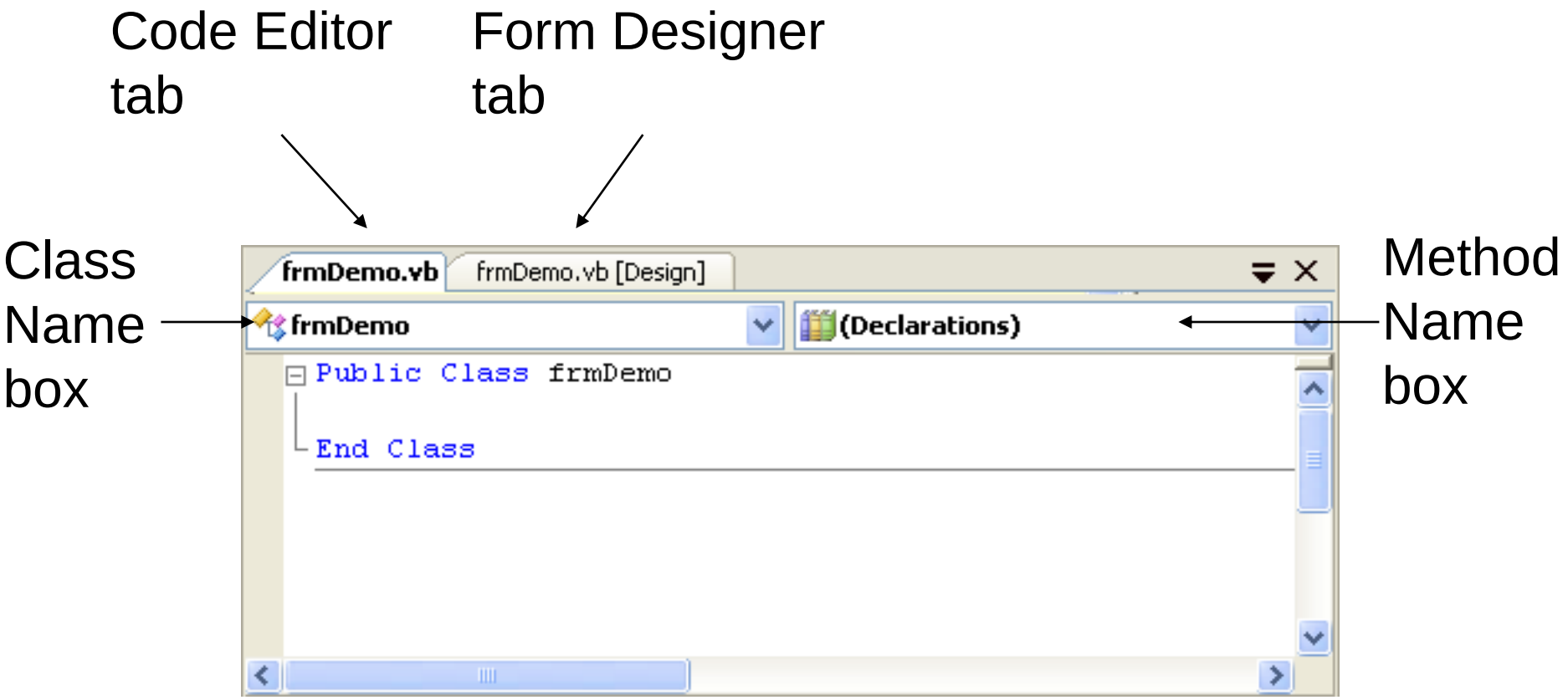
Positioning Controls



Aligning Controls



Code Editor


Code Editor tab
Form Designer tab
Class Name box
Method Name box
frmDemo.vb
frmDemo.vb [Design]
frmDemo
(Declarations)
Public Class frmDemo
End Class

Sample Code

```
Public Class frmDemo
  Private Sub txtFirst_TextChanged(...)
              Handles txtFirst.TextChanged
    txtFirst.ForeColor = Color.Blue
  End Sub
End Class
```

Chapter 3

Variables, Input, and Output

- 3.1 Numbers
- 3.2 Strings
- 3.3 Input and Output

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

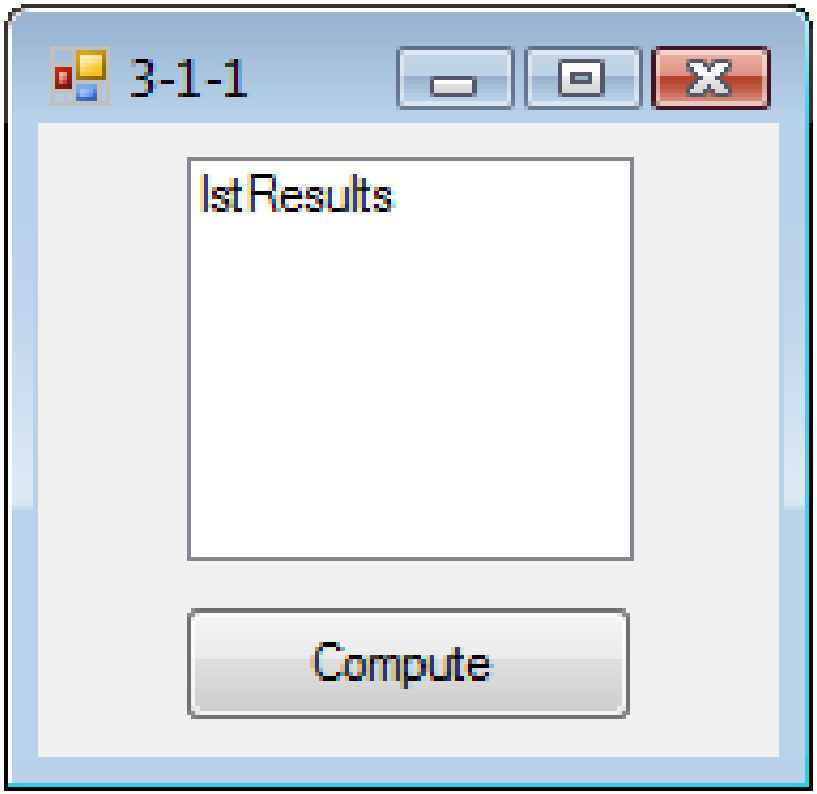
Let *n* be a number or a numeric expression.

What does the statement

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

do?

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
```

What is the result?

Numeric Variable

A *numeric variable* is a name to which a number can be assigned.

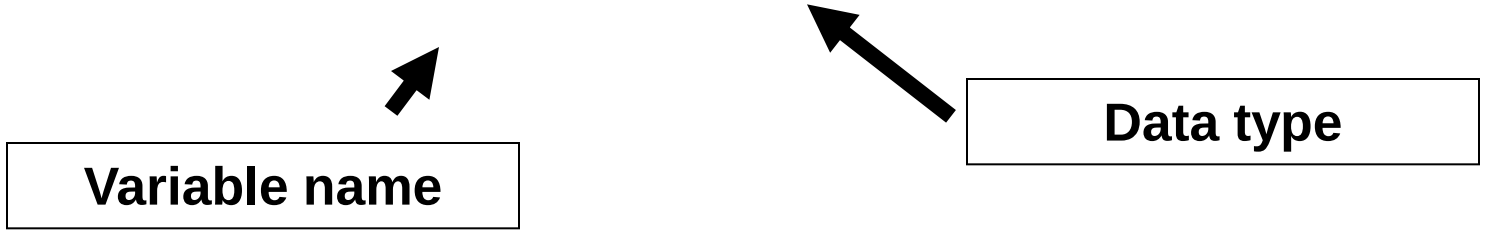
Examples:

speed
distance
interestRate
balance

VARIABLES

- Declaration:

```
Dim speed As Double
```



- Assignment:

```
speed = 50
```

VARIABLES

Visual Basic type	structure Storage size	Value range
Boolean	4 bytes	True or False
Byte	1 byte	0 to 255 (unsigned)
Char	2 bytes	0 to 65535 (unsigned)
Date	8 bytes	January 1, 1 CE to December 31, 9999
Decimal	12 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point;
Double	8 bytes	-1.797693134862 31E308 to -4.940656458412 47E-324 for negative values; 4.940656458412 47E-324 to 1.797693134862 32E308 for positive values

VARIABLES

Visual Basic type	structure Storage size	Value range
Integer	4 bytes	-2,147,483,648 to 2,147,483,647
Long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Object	4 bytes	Any type can be stored in a variable of type Object
Short	2 bytes	-32,768 to 32,767
Single	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
String	10 bytes + (2 * string length)	0 to approximately two billion Unicode characters

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

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

Assignment Statement

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

INCREMENTING

- To add 1 to the numeric variable *var*

```
var = var + 1
```

- Or as a shortcut

```
var += 1
```

- Or as a generalization

```
var += numeric expression
```

Built-in Functions

- Functions *return* a value

  Math.Sqrt(9) returns 3

  Int(9.7) returns 9

  Math.Round(2.7) is 3

Integer Data Type

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

- The statement

```
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
```

PARENTHESES

- Parentheses should be used liberally in numeric expressions

- In the absence of parentheses, the operations are carried out in the following order:

  ^, * and /, + and -

Three Types of Errors

- Syntax error
- Run-time error
- Logic error

Some Types of Syntax Errors

- Misspellings

```
lstBox.Itms.Add(3)
```

- Omissions

```
lstBox.Items.Add(2 + )
```

- Incorrect punctuation

```
Dim m; n As Integer
```

Displayed as blue underline in VS

A Type of Run-time Error

```
Dim numVar As Integer = 1000000
numVar = numVar * numVar
```

What's wrong with the above?

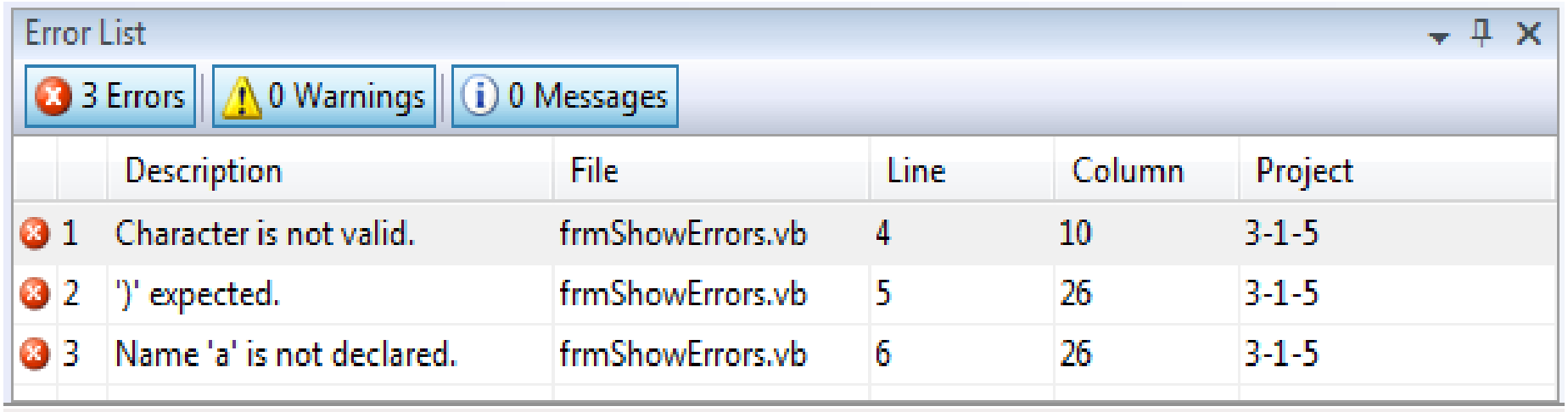
A Logical Error

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

What's wrong with the above?

Error List Window

- `Dim m; n As Double`
- `lstResults.Items.Add(5`
- `lstResults.Items.Add(a)`



– Variables, Input, and Output

- 3.1 Numbers
- 3.2 Strings
- 3.3 Input and Output

String Literal

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

Examples:

```
"hello"
"123-45-6789"
"#ab cde?"
```

String Literal

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

Examples:

Does this work?

```
"She said: "I'm tired.""
```

String Variable

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

Examples:

country

ssn

word

firstName

String Variable

- Declaration:

```
Dim firstName As String
```

- Assignment:

```
firstName = "Fred"
```

String Variable

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,

```
lstBox.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

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

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
```

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

Text Boxes for Input & Output

- The contents of a text box is always a string

- Input example

```
strVar = txtBox.Text
```

- Output example

```
txtBox.Text = strVar
```

Data Conversion

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

```
dblVar = CDbl(txtBox.Text)
```

Converts a String to a Double

```
txtBox.Text = CStr(numVar)
```

Converts a number to a string

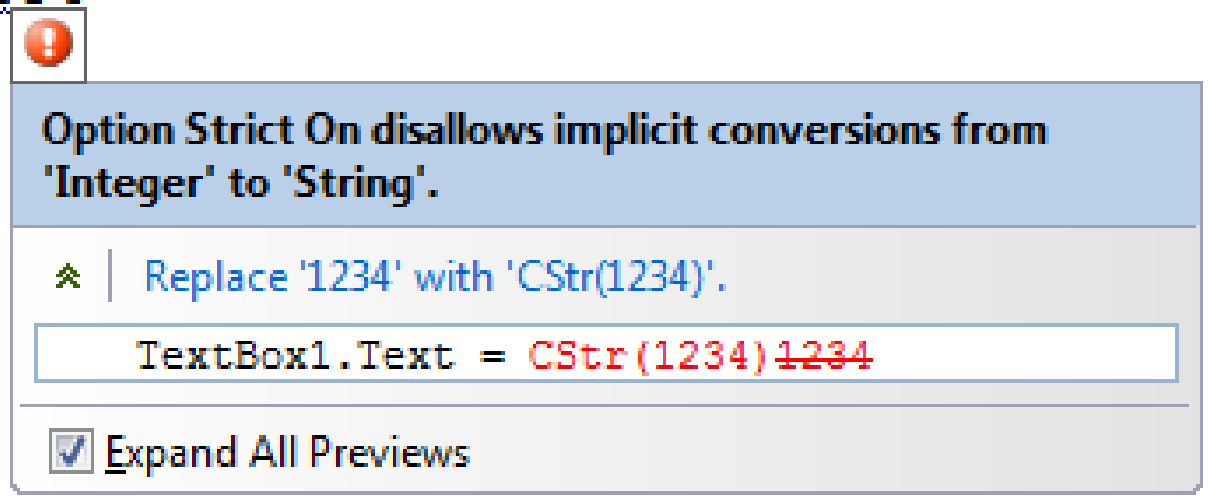
Widening and Narrowing

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

- Narrowing: assigning a Double value to an Integer variable
- Narrowing might not work. (Not every Double is an Integer.)
- Narrowing requires Cint.
  - Will loose information (everything after the decimal place)

Auto Correction

```
TextBox1.Text = 1234
```



With Option Strict On

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

Not Valid:	*Replace with:*
`intVar = dblVar`	`intVar = CInt(dblVar)`
`dblVar = strVar`	`dblVar = CDbl(strVar)`
`strVar = intVar`	`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"
```

Displays

```
We'll always have Paris. - Humphrey Bogart
```

Appending

- To append *str* to the string variable *var*

```
var = var & str
```

- Or as a shortcut

```
var &= str
```

Appending Example

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

String Properties and Methods

"Visual".Length is 6.

.length calculates the length of the string.

Varname = “blah”

Varname.length

String Properties and Methods

"Visual".ToUpper is VISUAL

.ToUpper makes everything upper case.

Varname = “blah”

String Properties and Methods

"123 Hike".ToLower is “123 hike”

.ToLower makes everything lower case

Varname = “Blah”

String Properties and Methods

"a" & " bcd ".Trim & "efg" is “abcdefg”

.trim removes leading/trailing spaces

Varname = “ blah “

Varname.trim

String properties

- Can apply a method onto a method
- What does this do?

```
Dim varname As String = "Tim Hortons"
varname.ToUpper.Replace("I", "O").ToLower()
```

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

Substring Method

Let *str* be a string

`str.Substring(`*`m`*`, `*`n`*`)`

is the substring of length *n*, beginning at position *m* in *str*

"Visual Basic".Substring(2, 3) ?

"Visual Basic".Substring(0, 1) ?

IndexOf Method

Let *str1* and *str2* be strings.

```
str1.IndexOf(str2)
```

is the position of the first occurrence of *str2* in *str1*

(**Note:** 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 `""` **(NOT `" "`)**, which contains 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

```
txtBox.Clear()
```

or the statement

```
txtBox.Text = ""
```

Initial Value of a String

- By default the initial value is **Nothing**
- Strings can be given a different initial value as follows:

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

COMMENTS

```
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
```

Internal Documentation

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 Character

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

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

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 in the event procedure in which they are declared

Scope

- 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***

  (Declarations section of Code Editor.)

Automatic Colorization

Comments – green

String literals – maroon

Keywords – blue

Note: Keywords are words such as Sub, Handles, Private, With, and End that have special meaning in Visual Basic. They cann

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Objec
        'this is my comment
        Dim varname As String = "Tim Hortons"
        varname.ToUpper.Replace("I", "O").ToLower()
    End Sub
End Class
```

Commenting

- Commenting is critical
  - For yourself and others
  - Have to do it right

COMMENTING

Code Commenting - This refers to writing descriptive variable names that are self explanatory. This is a minimalist form of commenting, and looks like this:

```
function addUserToDatabase(userName, userAge)
```

Without any additional information, you can tell that the function will add a user's name and age to a database. A common implementation of this is called Hungarian Notation.

COMMENTING

Inline Commenting - Specifically, these types of comments come at the end of a line of code, but we can also use this term to refer to comments inside of a function as well. This is the most basic form of commenting.

```
function calculateHitPoints(cUser) {
    var nStrength = document.getElementById("enemyStrength").value; //

    // subtract user size : small = 1, medium = 2, large = 3
    var nDamage = (nStrength * 3) ï¿½ cUser.nSize;
    return cUser.nCurrentHitPoints ï¿½ nDamage;
}
```

COMMENTING

Function Commenting - This type of commenting is found on the lines above a function, and reveals all of the necessary details about that function. This includes parameters, return values, and any logic quirks or decisions that were made:

```
/*
 * Summary:      Calculate hitpoints after attack using formula
 *               new = current ï¿½ ((enemyStrength*3) ï¿½ size)
 * Parameters:   cUser ï¿½ object containing hero's stats
 * Return:       Boolean indicating life or death
 */
function calculateHitPoints(cUser) {
    ï¿½
} // end calculateHitPoints
```

COMMENTING

Class / Page Commenting - Comments that refer to an entire page or top level object fall into this category. Usually these comments include a broad overview, last edit date, associated files, author, and contact information. Additionally, this may include a general footer at the bottom of every page. Kevin wrote some great templates for building these types of comments in his feature on using XHTML templates.

```
/*
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Title :
Author :
URL :

Description :

Created :
Modified :

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/
```

– Variables, Input, and Output

- 3.1 Numbers
- 3.2 Strings
- 3.3 Input and Output

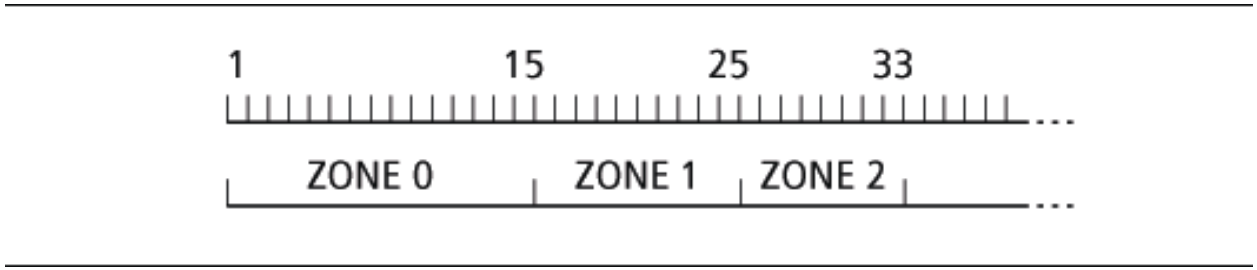
Formatting Output with Functions

Function	String Value
FormatNumber(12345.628, 1)	12,345.6
FormatCurrency(12345.628, 2)	$12,345.63
FormatPercent(0.183, 0)	18%

Formatting Output with Zones

- Use a fixed-width font such as Courier New
- Divide the characters into zones with a format string.

```
Dim fmtStr As String = "{0, 15}{1, 10}{2, 8}"
lstOutput.Items.Add(String.Format(fmtStr, _
                                  data0, data1, data2))
```



Formatting Output with Zones

- Use a fixed-width font such as Courier New
- Divide the characters into zones with a format string.

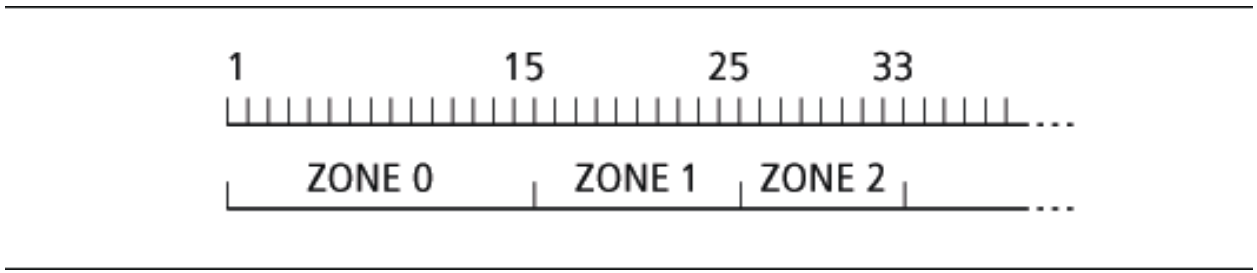
```
Dim fmtStr As String = "{0, 15}{1, 10}{2, 8}"
Debug.Print(String.Format(fmtStr, "abc",
  "def", "ghi"))
```

→ “ abc def ghi”

FORMATTING OUTPUT WITH ZONES

```
Dim fmtStr As String = "{0, -15}{1, 10}{2, 8}"
lstOutput.Items.Add(String.Format(fmtStr, _
                                  data0, data1, data2))
```

Here, 15 was preceded by a minus sign. This produces left justification in 0^{th} zone. There will be right justification in the other two zones.



Formatting Output with Zones

- Use a fixed-width font such as Courier New
- Divide the characters into zones with a format string.

```
Dim fmtStr As String = "{0,-15}{1, 10}{2, 8}"
Debug.Print(String.Format(fmtStr, "abc",
  "def", "ghi"))
```

- → “abc def ghi”

Reading Data from Files

- Data can be stored in text files and accessed with a StreamReader object.
- We assume that the text files have one piece of data per line.

Sample File: PAYROLL.TXT

Mike Jones ← Name
9.35 ← Hourly wage
35 ← Number of hours worked
John Smith
10.75
33

Steps to Use StreamReader

Execute a statement of the form

```
Dim readerVar As IO.StreamReader  = _
                IO.File.OpenText(filespec)
```

or the pair of statements

```
Dim readerVar As IO.StreamReader
readerVar = IO.File.OpenText(filespec)
```

Steps to Use StreamReader

Read items of data in order, one at a time, from the file with the ReadLine method.

```
strVar = readerVar.ReadLine
```

After the desired items have been read from the file, terminate the communications link

```
readerVar.Close()
```

EXAMPLE USING STREAMREADER

Mike Jones
9.35
35
John Smith
10.75
33

```
Dim name As String
Dim wage, hours As Double
Dim sr As IO.StreamReader  = _
                IO.File.OpenText("PAYROLL.TXT")
name = sr.ReadLine
wage = CDbl(sr.ReadLine)
hours = CDbl(sr.ReadLine)
lstBox.Items.Add(name & ": " & wage * hours)
```

OUTPUT: `Mike Jones: 327.25`

Comment on Example

Consider

```
lstBox.Items.Add(name & ": " & wage * hours)
```

The ampersand automatically converted **`wage * hours`** into a string before concatenating.

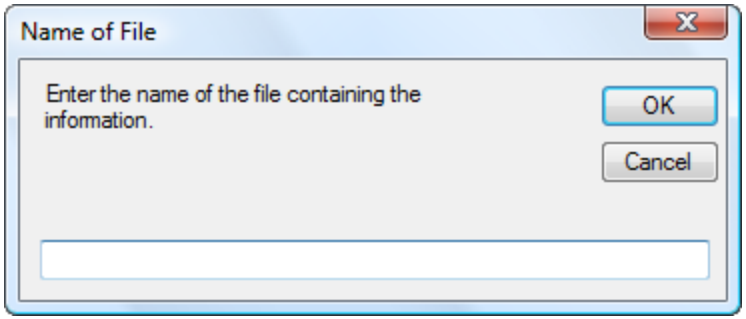
We didn't have to convert **`wage * hours`** with CStr.

Getting Input from an Input Dialog

```
stringVar = InputBox(prompt, title)
fileName = InputBox("Enter the name " _
    & "of the file containing the " & _
      "information.", "Name of File")
```

Title →

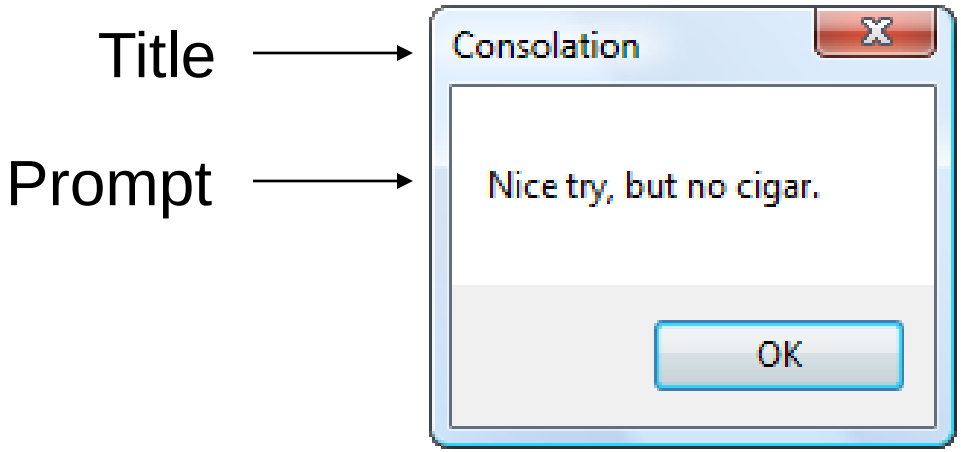
Prompt →



Using a Message Box for Output

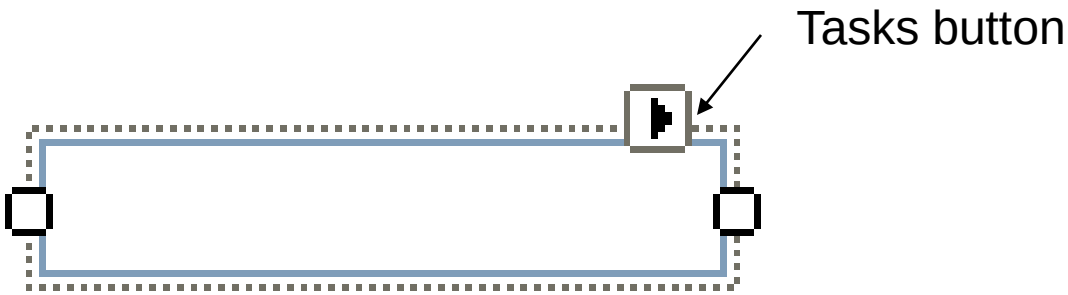
```
MessageBox.Show(prompt, title)

MessageBox.Show("Nice try, but no
  cigar.",  _
  "Consolation")
```



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.

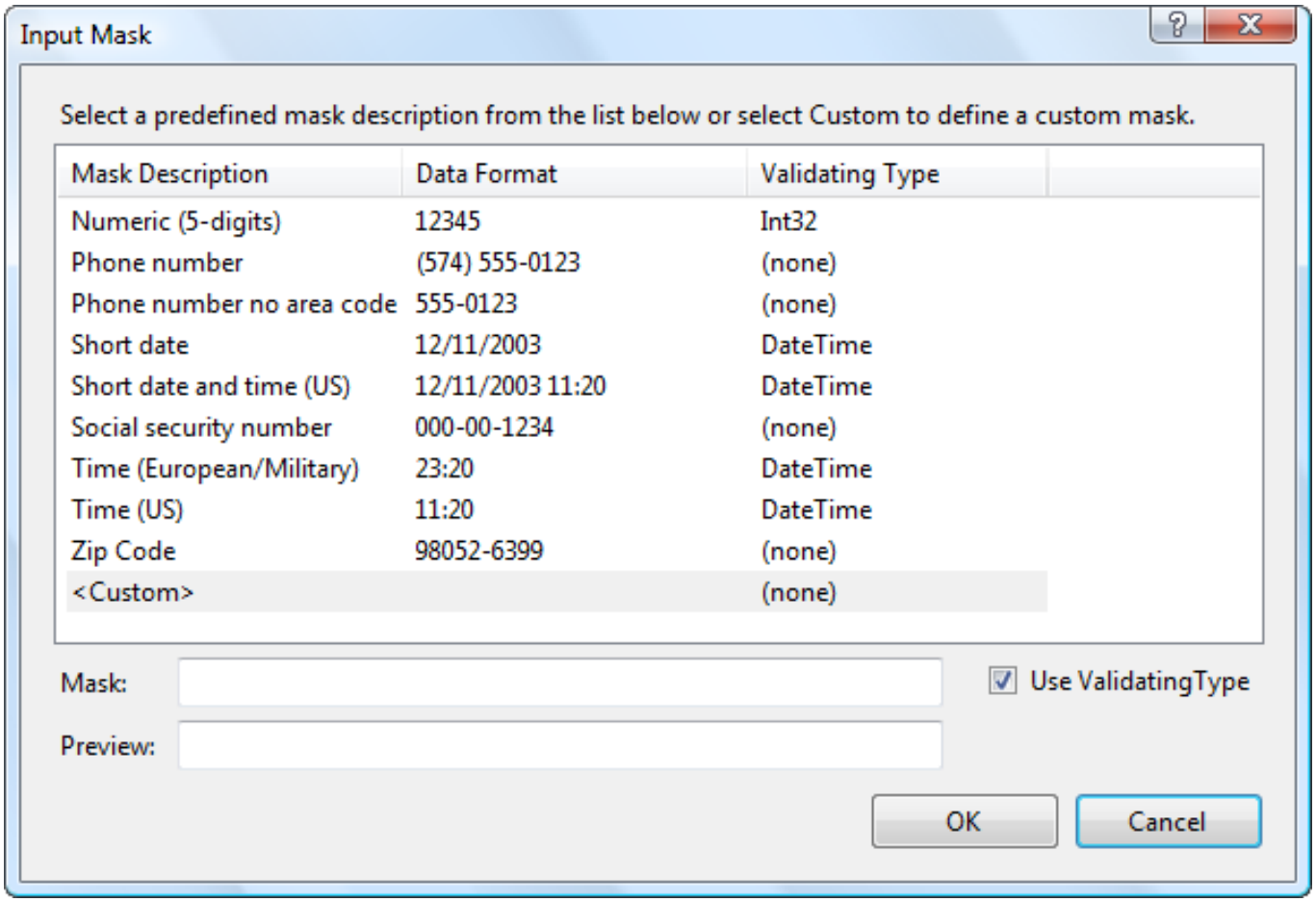


Masked Text Box Control

Click the Tasks button to reveal Set Mask property.

Click Set Mask to invoke Input Mask dialog box.

Input Mask Dialog Box



Mask

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: &&&&&&