Chapter 24 - VBScript

Outline

24.2 Operators

VBScript operation	Arithmetic operator	Algebraic expression	VBScript expression
Addition	+	$x + y$	x + y
Subtraction	-	$z - 8$	z - 8
Multiplication	*	yb	y * b
Division (floating-point)	/	$v \quad u$ or <Anchor0>	v / u
Division (integer)	\	none	v \ u
Exponentiation	^	q^{p}	q ^ p
Negation	-	$-e$	-e
Modulus	Mod	$q \bmod r$	q Mod r

Fig. 24.1 Arithmetic operators.

24.2 Operators

Standard algebraic equality operator or relational operator	VBScript comparison operator	Example of VBScript condition	Meaning of VBScript condition
=	=	d = g	d is equal to g
	<>	s <> r	s is not equal to r
>	>	y > x	y is greater than x
<	<	p < m	p is less than m
	>=	c >= z	c is greater than or equal to z
□ ’	<=	m <= s	m is less than or equal to s

Fig. 24.2 Comparison operators.

Truth tables for VBScript Logical Operators	
Logical And: True And True = True True And False = False False And True = False False And False = False	***Logical Or:*** True Or True = True True Or False = True False Or True = True False Or False = False
Logical Imp: True Imp True = True True Imp False = False False Imp True = True False Imp False = True	***Logical Eqv:*** True Eqv True = True True Eqv False = False False Eqv True = False False Eqv False = True
Logical Xor: True Xor True = False True Xor False = True False Xor True = True False Xor False = False	***Logical Not:*** Not True = False Not False = True

Fig. 24.3 Truth tables for VBScript logical operators.

24.3 Data Types and Control Structures

Subtype	Range/Description
Boolean	True or False
Byte	Integer in the range 0 to 255
Currency	–922337203685477.5808 to 922337203685477.5807
Date/Time	1 January 100 to 31 December 9999 / 0:00:00 to 23:59:59.
Double	–1.79769313486232E308 to –4.94065645841247E–324 (negative) 4.94065645841247E–324 to 1.79769313486232E308 (positive)
Empty	Uninitialized. This value is 0 for numeric types (e.g., double), False for booleans and the *empty string* (i.e., "") for strings.
Integer	–32768 to 32767
Long	–2147483648 to 2147483647
Object	Any object type.
Single	–3.402823E38 to –1.401298E–45 (negative) 1.401298E–45 to 3.402823E38 (positive)
String	0 to ~2000000000 characters.

Fig. 24.4 Some VBScript variant subtypes.

24.3 Data Types and Control Structures

JavaScript Control Structure	VBScript Control Structure Equivalent
sequence	sequence
if	If/Then/End If
if/else	If/Then/Else/End If
while	While/Wend or Do While/Loop
for	For/Next
do/while	Do/Loop While
switch	Select Case/End Select
none	Do Until/Loop
none	Do/Loop Until

Fig. 24.5 Comparing VBScript control structures to JavaScript control structures.

JavaScript

```
1  if ( s == t )
2     u = s + t;
3  else if ( s > t )
4     u = r;
5  else
6     u = n;
```

VBScript

```
1  If s = t Then
2     u = s + t
3  ElseIf s > t Then
4     u = r
5  Else
6     u = n
7  End If
```

Fig. 24.6 Comparing JavaScript's **if** structure to VBScript's **If** structure.

24.3 Data Types and Control Structures

JavaScript

```
1  switch ( x ) {
2     case 1:
3        alert("1");
4        break;
5     case 2:
6        alert("2");
7        break;
8     default:
9        alert("?");
10 }
```

VBScript

```
1  Select Case x
2     Case 1
3        Call MsgBox("1")
4     Case 2
5        Call MsgBox("2")
6     Case Else
7        Call MsgBox("?")
8  End Select
```

Fig. 24.7 Comparing JavaScript's **switch** with VBScript's **Select Case**.

24.3 Data Types and Control Structures

JavaScript	VBScript
1 `while ( !( x == 10 ) )` 2 `++x;`	1 `Do Until x = 10` 2 `x = x + 1` 3 `Loop`

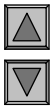
Fig. 24.8 Comparing JavaScript's **while** to VBScript's **Do Until**.

JavaScript	VBScript
1 `do {` 2 `++x;` 3 `} while ( !( x == 10 ) );`	1 `Do` 2 `x = x + 1` 3 `Loop Until x = 10`

Fig. 24.9 Comparing JavaScript's **do/while** to VBScript's **Do Loop/Until**.

JavaScript	VBScript
1 `x = 8;` 2 `for ( y = 1; y < x; y++ )` 3 `x /= 2;`	1 `x = 8` 2 `For y = 1 To x` 3 `x = x \ 2` 4 `Next`

Fig. 24.10 Comparing JavaScript's **for** to VBScript's **For**.



Outline

```
1   ' VBScript
2   For y = 2 To 20 Step 2
3      Call MsgBox( "y = " & y )
4   Next
```

Fig. 24.11 Using keyword Step in VBScript's For repetition structure.

For repetition structure with keyword **Step**

24.4 VBScript Functions

Function	Variant subtype returned	Description
`IsArray`	Boolean	Returns True if the variant subtype is an array and False otherwise.
`IsDate`	Boolean	Returns True if the variant subtype is a date or time and False otherwise.
`IsEmpty`	Boolean	Returns True if the variant subtype is Empty (i.e., has not been explicitly initialized by the programmer) and False otherwise.
`IsNumeric`	Boolean	Returns True if the variant subtype is numeric and False otherwise.
`IsObject`	Boolean	Returns True if the variant subtype is an object and False otherwise.
`TypeName`	String	Returns a string that provides subtype information. Some strings returned are "Byte", "Integer", "Long", "Single", "Double", "Date", "Currency", "String", "Boolean" and "Empty".
`VarType`	Integer	Returns a value indicating the subtype (e.g., 0 for Empty, 2 for integer, 3 for long, 4 for single, 5 for double, 6 for currency, 7 for date/time, 8 for string, 9 for object, etc.).

Fig. 24.12 Some variant functions.

24.4 VBScript Functions

Function	Description	Example
`Abs(x)`	Absolute value of `x`	`Abs(-7)` is 7 `Abs(0)` is 0 `Abs(76)` is 76
`Atn(x)`	Trigonometric arctangent of `x` (in radians)	`Atn(1)*4` is 3.141592653589793
`Cos(x)`	Trigonometric cosine of `x` (in radians)	`Cos(0)` is 1
`Exp(x)`	Exponential function e^x	`Exp(1.0)` is 2.71828 `Exp(2.0)` is 7.38906
`Int(x)`	Returns the whole-number part of `x`. `Int` rounds to the next smallest number.	`Int(-5.3)` is −6 `Int(0.893)` is 0 `Int(76.45)` is 76
`Fix(x)`	Returns the whole-number part of `x` [*Note*: `Fix` and `Int` are different. When `x` is negative, `Int` rounds to the next smallest number, while `Fix` rounds to the next-largest number.]	`Fix(-5.3)` is −5 `Fix(0.893)` is 0 `Fix(76.45)` is 76
`Log(x)`	Natural logarithm of `x` (base e)	`Log(2.718282)` is 1.0 `Log(7.389056)` is 2.0

`Rnd()`	Returns a pseudo-random floating-point number in the range 0 ≤ `Rnd` < 1. Call function `Randomize` once before calling `Rnd` to get a different sequence of random numbers each time the program is run.	`Call Randomize`
		`...`
		`z = Rnd()`

Fig. 24.13 VBScript math functions.

24.4 VBScript Functions

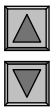
Function	Description	Example
`Round(x, y)`	Rounds `x` to `y` decimal places. If `y` is omitted, `x` is returned as an integer.	`Round(4.844)` is 5 `Round(5.7839, 2)` is 5.78
`Sgn(x)`	Sign of `x`	`Sgn(-1988)` is –1 `Sgn(0)` is 0 `Sgn(3.3)` is 1
`Sin(x)`	Trigonometric sine of `x` (in radians)	`Sin(0)` is 0
`Sqr(x)`	Square root of `x`	`Sqr(900.0)` is 30.0 `Sqr(9.0)` is 3.0
`Tan(x)`	Trigonometric tangent of `x` (in radians)	`Tan(0)` is 0

Fig. 24.13 VBScript math functions.

24.4 VBScript Functions

Function	Description
`FormatCurrency`	Returns a string formatted according to the local machine's currency **Regional Settings** (in the **Control Panel**). For example, the call `FormatCurrency("-1234.789")` returns "($1,234.79)" and the call `FormatCurrency(123456.789)` returns "$123,456.79". Note the rounding to the right of the decimal place.
`FormatDateTime`	Returns a string formatted according to the local machine's date/time **Regional Settings** (in the **Control Panel**). For example, the call `FormatDateTime(Now, vbLongDate)` returns the current date in the format "Wednesday, September 01, 1999" and the call `FormatDateTime(Now, vbShortTime)` returns the current time in the format "17:26". Function ***Now*** returns the local machine's time and date. Constant vbLongDate indicates that the day of the week, month, day and year is displayed. Constant vbShortTime indicates that the time is displayed in 24-hour format. Consult the VBScript documentation for additional constants that specify other date and time formats.
`FormatNumber`	Returns a string formatted according to the number **Regional Settings** (in the **Control Panel**) on the local machine. For example, the call `FormatNumber("3472435")` returns "3,472,435.00" and the call `FormatNumber(-123456.789)` returns "-123,456.79". Note the rounding to the right of the decimal place.
`FormatPercent`	Returns a string formatted as a percentage. For example the call `FormatPercent(".789")` returns "78.90%" and the call `FormatPercent(0.45)` returns "45.00%".

Fig. 24.14 Some VBScript formatting functions.



Outline

Addition.html

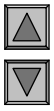
```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5   <!--Fig. 24.15: addition.html -->
6   <!--Adding Integers           -->
7
8   <html xmlns = "http://www.w3.org/1999/xhtml">
9      <head>
10        <title>Our first VBScript</title>
11
12        <script type = "text/vbscript">
13           <!--
14           Option Explicit
15           Dim intTotal
16
17           Sub cmdAdd_OnClick()
18              Dim intValue
19
20              intValue = InputBox(_
21                 "Enter an integer", "Input Box", , 1000, 1000)
22              intTotal = CInt( intTotal ) + CInt( intValue )
23              Call MsgBox("You entered " & intValue & _
24                 "; total so far is " & intTotal, , "Results")
25           End Sub
26           -->
27        </script>
28     </head>
29
30     <body>
31        Click the button to add an integer to the total.
32        <hr />
33        <form action = "">
34        <input name = "cmdAdd" type = "button"
35               value = "Click Here to Add to the Total" />
```

Set **type** to **VBScript**.

Option Explicit statement.

Define procedure **OnClick** for the **cmdAdd** button.

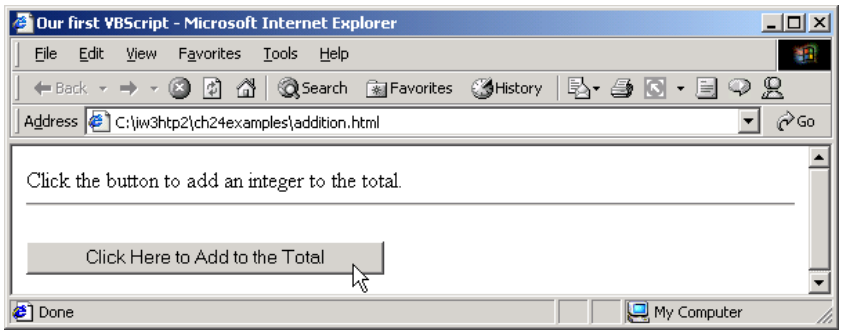
Use **CInt** to convert input values from string subtype to integer subtype.



Outline

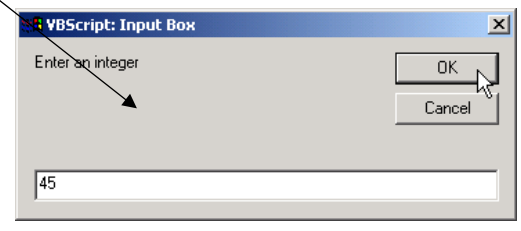
Addition.html

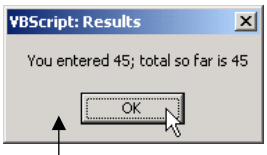
```
36        </form>
37     </body>
38  </html>
```



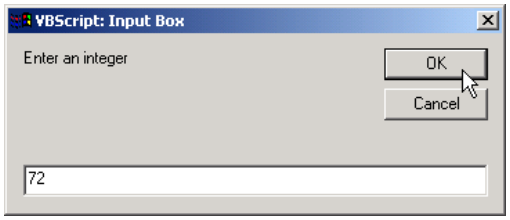
Program Output

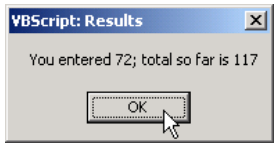
input dialog

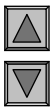




message dialog







Outline

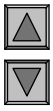
Site.html

```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5   <!-- Fig. 24.16: site.html -->
6   <!-- Displaying a Web site -->
7
8   <html xmlns = "http://www.w3.org/1999/xhtml">
9      <head>
10        <title>Select a site to browse</title>
11     </head>
12
13     <body>
14        Select a site to browse<p>
15        <hr />
16        <form action = "">
17        <select name = "SiteSelector" size = "1">
18
19           <option value = "http://www.deitel.com">
20               Deitel & Associates, Inc.
21           </option>
22
23           <option value = "http://www.prenhall.com">
24               Prentice Hall
25           </option>
26
27           <option value = "http://www.phptr.com/p
28               Prentice Hall Interactive
29           </option>
30
31        </select>
32
33        <!-- VBScript code -->
34        <script for = "SiteSelector" event = "onchange"
35                type = "text/vbscript">
```

Create **form** with **select** component.

The **event** attribute indicates the event to which the script responds (**OnChange**).

The **<script>** tag's **for** attribute indicates the XHTML component on which the script operates (**SiteSelector**).

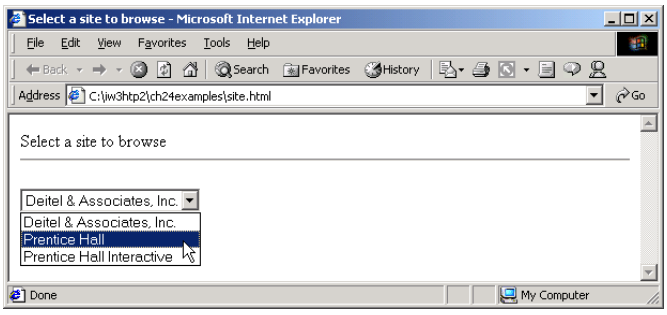


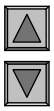
Outline

Site.html

```
36           <!--
37           Document.Location = Document.Forms( 0 ).SiteSelector.Value
38           -->
39        </script>
40        </form></p>
41     </body>
42  </html>
```

Program Output





Outline

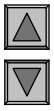
Minimum.html

```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5   <!--Fig. 24.17: minimum.html -->
6   <!-- VBScript Procedures     -->
7
8   <html xmlns = "http://www.w3.org/1999/xhtml">
9      <head>
10        <title>Using VBScript Procedures</title>
11
12        <script type = "text/vbscript">
13           <!--
14           Option Explicit
15
16           ' Find the minimum value. Assume that first value is
17           ' the smallest.
18           Function Minimum( min, a, b )
19
20              If a < min Then
21                 min = a
22              End If
23
24              If b < min Then
25                 min = b
26              End If
27
28              Minimum = min     ' Return value
29           End Function
30
31           Sub OddEven( n )
32              If n Mod 2 = 0 Then
33                 Call MsgBox( n & " is the smallest and is even" )
```

Define procedures **Minimum** and **OddEven**.

Single-line comment.

Use modulus operator to determine whether number odd or even.



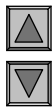
Outline

Minimum.html

```
34              Else
35                 Call MsgBox( n & " is the smallest and is odd" )
36              End If
37           End Sub
38
39           Sub cmdButton_OnClick()
40              Dim number1, number2, number3, smallest
41
42              ' Convert each input to Long subtype
43              number1 = CLng( Document.Forms( 0 ).txtBox1.Value )
44              number2 = CLng( Document.Forms( 0 ).txtBox2.Value )
45              number3 = CLng( Document.Forms( 0 ).txtBox3.Value )
46
47              smallest = Minimum( number1, number2, number3 )
48              Call OddEven( smallest )
49           End Sub
50           -->
51        </script>
52     </head>
53
54     <body>
55        <form action = ""> Enter a number
56           <input type = "text" name = "txtBox1" size = "5"
57              value = "0" />
58              <p>Enter a number
59           <input type = "text" name = "txtBox2" size = "5"
60              value = "0" /></p>
61              <p>Enter a number
62           <input type = "text" name = "txtBox3" size = "5"
63              value = "0" /></p>
64           <p><input type = "button" name = "cmdButton"
65              value = "Enter" /></p>
66
```

Define an event procedure for handling

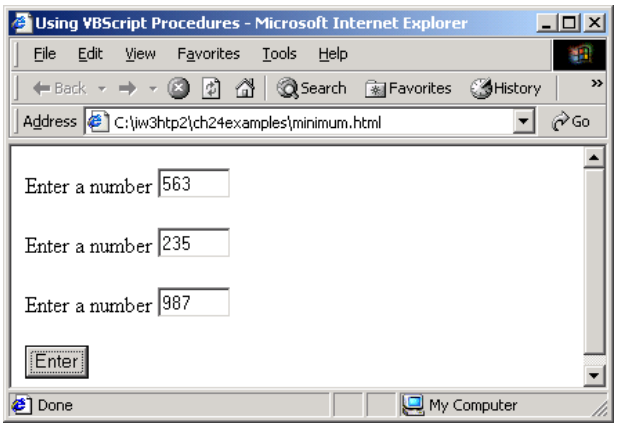
Pass the smallest number to procedure **OddEven**.



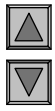
Outline

```
67          </form>
68       </body>
69    </html>
```

Minimum.html



Program Output



Outline

Arrays.html

```
1    <?xml version = "1.0"?>
2    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5    <!--Fig. 24.18: arrays.html -->
6    <!--VBScript Arrays         -->
7
8    <html xmlns = "http://www.w3.org/1999/xhtml">
9       <head>
10         <title>Using VBScript Arrays</title>
11
12         <script type = "text/vbscript">
13            <!--
14            Option Explicit
15
16            Public Sub DisplayArray( x, s )
17               Dim j
18
19               Document.Write( s & ": " )
20               For j = 0 To UBound( x )
21                  Document.Write( x( j ) & " " )
22               Next
23
24               Document.Write( "<br />" )
25            End Sub
26
27            Dim fixedSize( 3 ), fixedArray, dynamic(), k
28
29            ReDim dynamic( 3 )  ' Dynamically size array
30            fixedArray = Array( "A", "B", "C" )
31
32            ' Populate arrays with values
33            For k = 0 to UBound( fixedSize )
34               fixedSize( k ) = 50 - k
35               dynamic( k ) = Chr( 75 + k )
```

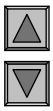
Define procedure **DisplayArray**.

Function **UBound** returns the upper bound (i.e., the highest-numbered index).

Initialize arrays.

Statement **ReDim** allocates memory for array **dynamic**.

Function **array** takes any number of arguments and returns an array containing those arguments.



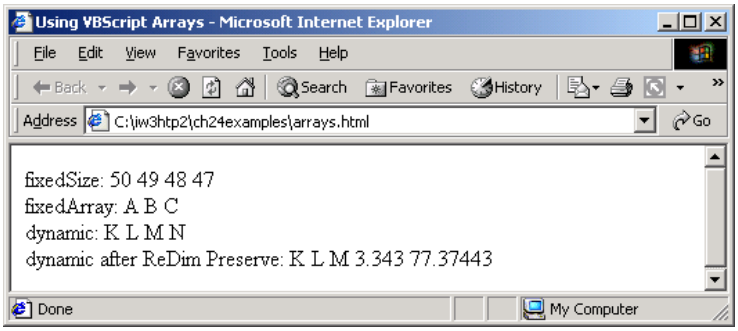
Outline

Arrays.html

```
36            Next
37
38            ' Display contents of arrays
39            Call DisplayArray( fixedSize, "fixedSize" )
40            Call DisplayArray( fixedArray, "fixedArray" )
41            Call DisplayArray( dynamic, "dynamic" )
42
43            ' Resize dynamic, preserve current values
44            ReDim Preserve dynamic( 5 )
45            dynamic( 3 ) = 3.343
46            dynamic( 4 ) = 77.37443
47
48            Call DisplayArray( dynamic, _
49                               "dynamic after ReDim Preserve" )
50            -->
51         </script>
52      </head><body></body>
53   </html>
```

Call procedure **DisplayArray**.

Reallocate **dynamic**'s memory to 5 elements. Keyword **Preserve**, when used with **ReDim**, maintains the current values in the array.



Program Output

24.7 String Manipulation

Function	Description
Asc	Returns the ASCII numeric value of a character. For example, **Asc** ("x") returns 120.
Chr	Returns the character representation for an ASCII value. For example the call **Chr** (120) returns "**x**." The argument passed must be in the range 0 to 255 inclusive, otherwise an error occurs.
InStr	Searches a string (i.e., the first argument) for a substring (i.e., the second argument). Searching is performed from left to right. If the substring is found, the index of the found substring in the search string is returned. For example, the call **Instr** ("sparrow", "arrow") returns 3 and the call **Instr** ("japan", "wax") returns 0.
Len	Returns the number of characters in a string. For example, the call **Len** ("hello") returns 5.
LCase	Returns a lowercase string. For example, the call **LCase** ("HELLO@97[") returns "**hello@97 [**."
UCase	Returns an uppercase string. For example, the call **UCase** ("hello@97[") returns "**HELLO@97 [**."
Left	Returns a string containing characters from the left side of a string argument. For example, the call **Left** ("Web", 2) returns "**We**."
Mid	Function **Mid** returns a string containing a range of characters from a string. For example, the call **Mid** ("abcd", 2, 3) returns "**bcd**."
Right	Returns a string containing characters from the right side of a string argument. For example, the call **Right** ("Web", 2) returns "**eb**."
Space	Returns a string of spaces. For example, the call **Space** (4) returns a string containing four spaces.

Fig. 24.19 Some string-manipulation functions.

24.7 String Manipulation

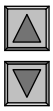
Function	Description
`StrComp`	Compares two strings for equality. Returns `1` if the first string is greater than the second string, returns `-1` if the first string is less than the second string and returns `0` if the strings are equivalent. The default is a binary comparison (i.e., case-sensitive). An optional third argument of `vbTextCompare` indicates a case-insensitive comparison. For example the call `StrComp("bcd", "BCD")` returns `1`, the call `StrComp("BCD", "bcd")` returns `-1`, the call `StrComp("bcd", "bcd")` returns `0` and the call `StrComp("bcd", "BCD", vbTextCompare)` returns `0`.
`String`	Returns a string containing a repeated character. For example, the call `String(4,"u")` returns "`uuuu`."
`Trim`	Returns a string that does not contain leading or trailing space characters. For example the call `Trim("  hi  ")` returns "`hi`."
`LTrim`	Returns a string that does not contain any leading space characters. For example, the call `LTrim("  yes")` returns "`yes`."
`RTrim`	Returns a string that does not contain any trailing space characters. For example, the call `RTrim("no  ")` returns "`no`".
`Filter`	Returns an array of strings containing the result of the `Filter` operation. For example, the call `Filter(Array("A","S","D","F","G","D"),"D")` returns a two-element array containing `"D"` and `"D"`, and the call `Filter(Array("A","S","D","F","G","D"),"D",False)` returns an array containing `"A"`, `"S"`, `"F"` and `"G"`.
`Join`	Returns a string containing the concatenation of array elements separated by a delimiter. For example, the call `Join(Array("one","two","three"))` returns "`one two three`." The default delimiter is a space which can be changed by passing a delimiter string for the second argument. For example, the call `Join(Array("one","two","three"),"$^")` returns "`one$^two$^three`."

Fig. 24.19 Some string-manipulation functions.

24.7 String Manipulation

Function	Description
`Replace`	Returns a string containing the results of a `Replace` operation. Function `Replace` requires three string arguments: the string where characters will be replaced, the substring to search for and the replacement string. For example, `Replace("It's Sunday and the sun is out","sun","moon")` returns "`It's Sunday and the moon is out`." Note the case-sensitive replacement.
`Split`	Returns an array containing substrings. The default delimiter for `Split` is a space character. For example, the call `Split("I met a traveller")` returns an array containing elements `"I"`, `"met"`, `"a"` and `"traveller"` and `Split("red,white,and  blue",",")` returns an array containing elements `"red"`, `"white"` and `"and  blue"`. The optional second argument changes the delimiter.
`StrReverse`	Returns a string in reverse order. For example, the call `StrReverse("deer")` returns "`reed`."
`InStrRev`	Searches a string (i.e., the first argument) for a substring (i.e., the second argument). Searching is performed from right to left. If the substring is found, the index of the found substring in the search string is returned. For example, the call `InstrRev("sparrow","arrow")` returns `3`, the call `InstrRev("japan","wax")` returns `0` and the call `InstrRev("to be or not to be","to be")` returns `14`.

Fig. 24.19 Some string-manipulation functions.



Outline

Piglatin.html

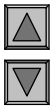
```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5   <!--Fig. 24.20: piglatin.html  -->
6   <!-- VBScript String Functions -->
7
8   <html xmlns = "http://www.w3.org/1999/xhtml">
9      <head>
10        <title>Using VBScript String Fu
11
12        <script type = "text/vbscript">
13           <!--
14           Option Explicit
15
16           Public Function TranslateToPigLatin( englishPhrase )
17              Dim words      ' Stores each individual word
18              Dim k, suffix
19
20              ' Get each word and store in words the
21              ' default delimiter for Split is a space
22              words = Split( englishPhrase )
23
24              For k = 0 To UBound( words )
25                 ' Check if first letter is a vowel
26                 If InStr( 1, "aeiou", _
27                       LCase( Left( words( k ), 1 ) ) ) Then
28                    suffix = "y"
29                 Else
30                    suffix = "ay"
31                 End If
32
```

Define **Function** procedure **TranslateToPigLatin**

Split phrase into words

Convert each word to pig Latin

Function **LCase** returns a lowercase string.



Outline

Piglatin.html

```
33                ' Convert the word to pig Latin
34                words( k ) = Right( words( k ), _
35                             Len( words( k ) ) - 1 ) & _
36                               Left( words( k ), 1 ) & suffix
37             Next
38
39             ' Return translated phras
40             ' is separated by spaces
41             TranslateToPigLatin = Joi
42          End Function
43
44          Sub cmdButton_OnClick()
45             Dim phrase
46
47             phrase = Document.Forms( 0 ).txtInput.Value
48
49             Document.forms( 0 ).txtPigLatin.Value = _
50                TranslateToPigLatin( phrase )
51          End Sub
52          -->
53       </script>
54    </head>
55
56    <body>
57       <form action = ""> Enter a sentence
58          <input type = "text" name = "txtInput" size = "50" />
59          <p>Pig Latin
60          <input type = "text" name = "txtPigLatin" size = "70" />
61          </p><p>
62          <input type = "button" name = "cmdButton"
63             value = "Translate" /></p>
64       </form>
65    </body>
66 </html>
```

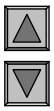
Function **Len** returns the number of

Function **Left** returns a string containing characters from the left side of a string argument.

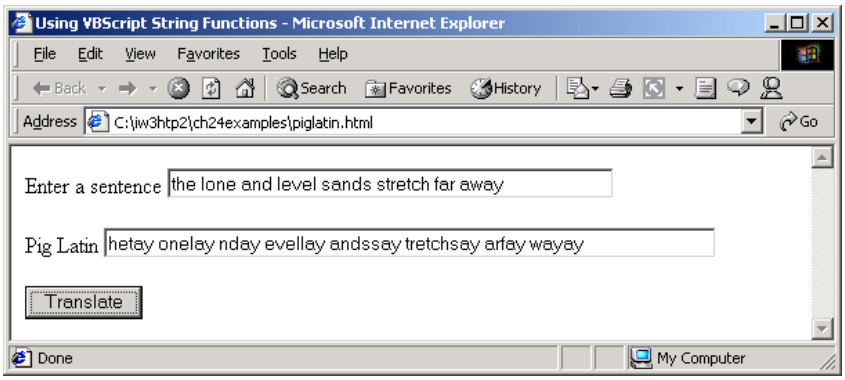
ment.

Return translated phrase using **Join** function

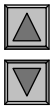
Define an event procedure for **cmdButton**'s **OnClick** event.



Outline



Program Output

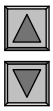


Outline

```
1    Private mHour
2
3    Public Property Let Hour( hr )
4       If hr >= 0 And hr < 24 Then
5          mHour = hr
6       Else
7          mHour = 0
8       End If
9    End Property
```

Fig. 24.21 Simple Property Let procedure.

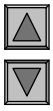
A simple **Property Let** procedure




```
1    Public Property Get Hour()
2       Hour = mHour
3    End Property
```

Fig. 24.22 Simple Property Get procedure.

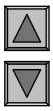
A simple **Property Get** procedure




```
1    Class CTime1
2       Private mHour
3
4       Public Property Let Hour( hr )
5          If hr >= 0 And hr < 24 Then
6             mHour = hr
7          Else
8             mHour = 0
9          End If
10      End Property
11
12      Public Property Get Hour()
13         Hour = mHour
14      End Property
15   End Class
```

A simple **Class** definition

Fig. 24.23 Simple Class definition.



Outline

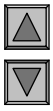
Classes.html

```
1    <?xml version = "1.0"?>
2    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5    <!--Fig. 24.24: classes.html -->
6    <!-- VBScript Class          -->
7
8    <html xmlns = "http://www.w3.org/1999/xhtml">
9       <head>
10         <title>Using a VBScript Class</title>
11
12         <script type = "text/vbscript">
13            <!--
14            Option Explicit
15
16            Class Person
17               Private name, yearsOld, ssn
18
19               Public Property Let FirstName( fn )
20                  name = fn
21               End Property
22
23               Public Property Get FirstName()
24                  FirstName = name
25               End Property
26
27               Public Property Let Age( a )
28                  yearsOld = a
29               End Property
30
31               Public Property Get Age()
32                  Age = yearsOld
33               End Property
34
35               Public Property Let SocialSecurityNumber( n )
```

Define **Class** Person

Define **Property Let** and **Property Get** procedures

Define **Property Let SocialSecurityNumber**



Outline

Classes.html

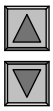
```
36
37                If Validate( n ) Then
38                   ssn = n
39                Else
40                   ssn = "000-00-0000"
41                   Call MsgBox( "Invalid Social Security Format" )
42                End If
43
44             End Property
45
46             Public Property Get SocialSecurityNumber()
47                SocialSecurityNumber = ssn
48             End Property
49
50             Private Function Validate( expression )
51                Dim regularExpression
52                Set regularExpression = New RegExp
53
54                regularExpression.Pattern = "^\d{3}-\d{2}-\d{4}$"
55
56                If regularExpression.Test( expression ) Then
57                   Validate = True
58                Else
59                   Validate = False
60                End If
61
62             End Function
63
64             Public Function ToString()
65                ToString = name & Space( 3 ) & age & Space( 3 ) _
66                      & ssn
67             End Function
68
69          End Class    ' Person
70
```

Call function **Validate**

Define **Function Validate**

Use regular expression to check format

Define **Function ToString**.



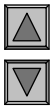
Outline

Classes.html

```
71          Sub cmdButton_OnClick()
72             Dim p                ' Declare object reference
73             Set p = New Person   ' Instantiate Person object
74
75             With p
76                .FirstName = Document.Forms(0).txtBox1.Value
77                .Age = CInt( Document.Forms(0).txtBox2.Value )
78                .SocialSecurityNumber =_
79                   Document.Forms(0).txtBox3.Value
80                Call MsgBox( .ToString() )
81             End With
82
83          End Sub
84          -->
85       </script>
86    </head>
87
88    <body>
89       <form action = "">Enter first name
90          <input type = "text" name = "txtBox1" size = "10" />
91          <p>Enter age
92          <input type = "text" name = "txtBox2" size = "5" /></p>
93          <p>Enter social security number
94          <input type = "text" name = "txtBox3" size = "10" />
95          </p><p>
96          <input type = "button" name = "cmdButton"
97             value = "Enter" /></p>
98       </form>
99    </body>
100 </html>
```

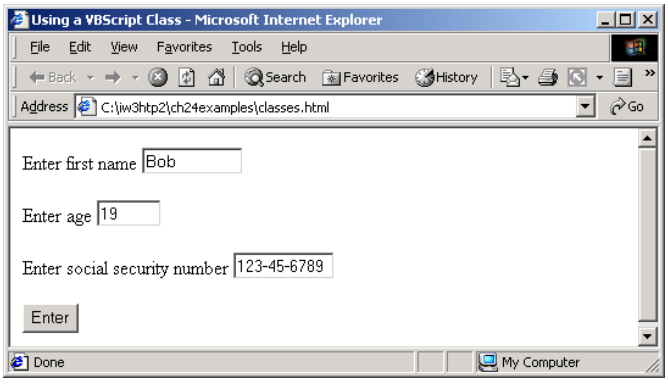
Provide an event procedure for

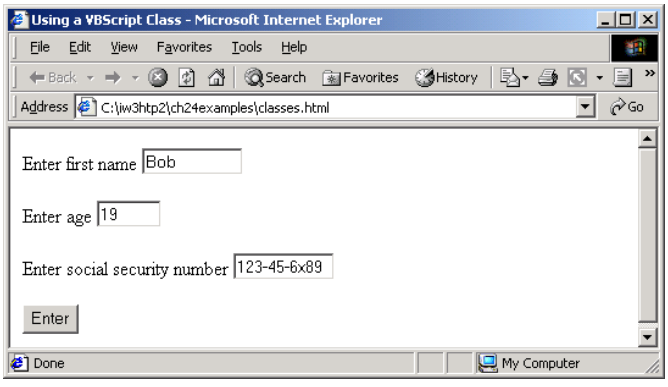
Use the **With/End With** statement to set several property values for **p** and call **p**'s **ToString** method.

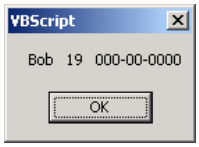


Outline

Program Output







24.9 Operator Precedence Chart

Operator	Type	Associativity
`()`	parentheses	left to right
`-`	unary minus	left to right
`^`	exponentiation	left to right

`*`
`/`
`\`
	multiplication
	division
	integer division
		left to right

`Mod`	modulus	left to right
`+` `-`	addition subtraction	left to right
`&`	string concatenation	left to right

`=`
`<>`
`<`
`<=`
`>`
`>=`
`Is`
	equality
	inequality
	less than
	less than or equal
	greater than
	greater than or equal
	object equivalence
		left to right

`Not`	logical NOT	left to right
`And`	logical AND	left to right
`Or`	logical OR	left to right
`Xor`	logical exclusive OR	left to right
`Eqv`	logical equivalence	left to right
`Imp`	logical implication	left to right

Fig. 24.25 VBScript operator precedence chart.