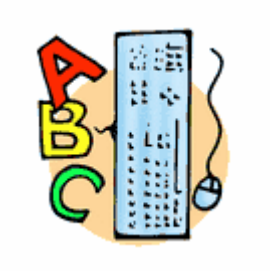
WORKING WITH STRINGS

Visual Basic provides a number of functions that make it easy to work with Strings. Before we take a look at some of these functions, it is important to understand how the characters in a string are stored in memory.



COMPARING STRINGS

Strings, like numbers, can be compared using relational operators. For example, if I wanted a program to check if two words are the same, I could compare the strings as follows:

```
If word1 = word2 Then
      MsgBox("The words are the same!")
Else
      MsgBox("The words are not the same!")
End If
```

It is important to note that computers do not store characters in memory; they store numeric codes that represent characters. Visual Basic uses something called **Unicode**, which is a set of numbers that represent all the letters of the alphabet (uppercase and lowercase), the numbers 0 to 9, punctuation symbols, and special characters. Each character is stored in memory as its corresponding Unicode number.

In Unicode, letters are arranged alphabetically. Therefore, since **A** comes before **B**, the numeric code for the letter **A** is less than the code for the letter **B**. So if we were to compare the String "Algeria" with the String "Barcelona" with the following line of code,

```
If "Algeria" > "Barcelona" Then
      MsgBox("Algeria comes before Barcelona!")
Else
      MsgBox("Barcelona comes before Algeria!")
End If
```

the statement *Algeria comes before Barcelona!* would be returned because the program would see that the value of **A** is less than the value of **B**.

Uppercase letters come before lowercase letters, so the numeric code for **B** is less than the numeric code for **a**.

When you use relational operators to compare strings, the strings are compared character by character. So, for example, if we compare the name "Mary" with the name "Mark", "Mark" would be less than "Mary" because although the letters "M", "a" and "r" are identical, the letter "k" in "Mark" is less than the letter "y" in "Mary".

USING THE *Compare()* METHOD

A program will often need to compare String objects. For example, when you log into your hotmail account, the program needs to check if the e-mail address you entered matches the e-mail address in its database. Relational operators (=, >, <, etc.) can be used to compare strings, however, they use

Unicode values of the strings to determine the relationship between the strings. This can produce unexpected results because uppercase and lowercase letters have different values. For example, the letter “r” and the letter “R” have two different Unicode values.

When comparing strings, therefore, the **Compare()** method should be used. The **Compare()** method takes the following format:

```
String.Compare(string1, string2, case-insensitive)
```

The **Compare()** method requires the two strings that will be compared against each other as well as a **Boolean** value which determines whether it should take into account case-sensitivity. If set to true, then it will not account for case-sensitivity; if false, then it will account for case-sensitivity. If **string1** and **string2** are the same, the method returns **0**. If **string1** is greater than **string2**, a positive number is returned. If **string1** is less than **string2**, a negative number is returned.

In the following example, we will compare the String “bulhao” with the String “Bulhao”:

```
Dim name As String = "bulhao"
Dim newName As String = "Bulhao"

If String.Compare(name, newName, True) = 0 Then
      MsgBox("The same!", MsgBoxStyle.Exclamation)
Else
      MsgBox("Not the same!", MsgBoxStyle.Critical)
End If
```

THE Like OPERATOR

The **Like** operator is another way to compare String objects, the difference being that it can be used to perform **pattern matching**. Pattern matching allows wildcard characters, character lists, and character ranges to match strings. The **Like** operator takes the following form:

```
result = string Like pattern
```

The variable **result** is a Boolean that is assigned the value **True** if the String variable matches the **pattern**, otherwise it returns **False**. A pattern can take any of the following forms:

PATTERN	DESCRIPTION	EXAMPLE

?	Used in place of any single character.	`Dim word As String = “Run”`
		`Dim pattern As String = “?un”`
		`lblOutput.Text = word Like pattern`

*	Used in place of many characters.	`Dim word As String = “Visual Basic”`
		`Dim pattern As String = “Visual *”`
		`lblOutput.Text = word Like pattern`

#	Used in place of a single number.	`Dim word As String = “Case 9876”`
		`Dim pattern As String = “Case 987#”`
		`lblOutput.Text = word Like pattern`

[]	Used to enclose a list of characters.	`Dim word As String = "B"`
		`Dim pattern As String = "[A, B, C, D, E]"`
		`lblOutput.Text = word Like pattern`

-	Used to indicate a range of characters in a character list.	`Dim word As String = "B"`
		`Dim pattern As String = "[A-F]"`
		`lblOutput.Text = word Like pattern`

In each of the above examples, the **Like** operator returns **True**.

STRING FUNCTIONS

The following table outlines some of the methods included in Visual Basic's **String** class. For the examples listed below assume that we have a **String** variable called **word** which equals the word "Hello".

FUNCTION	DESCRIPTION	FORMAT	EXAMPLE	RESULT
Chars	• Returns the character at a specified position in a string	StringValue.**Chars** (index As Integer)	word.Chars(1)	Returns **e**
Concat	• Concatenates two strings	String.**Concat** (x As String, y As String)	String.Concat (word, " everyone!")	Returns **Hello everyone!**
Contains	• Returns a boolean value indicating whether the specified string is contained within another string	StringValue.**Contains** (x As String)	word.Contains ("o")	Returns **True**
EndsWith	• Returns a boolean value indicating whether the end of this instance matches the specified string	StringValue.**EndsWith** (x As String)	word.EndsWith ("ol")	Returns **False**
ToUpper	• Converts a string value to uppercase.	StringValue.**ToUpper()**	word.ToUpper()	Returns **HELLO**
ToLower	• Converts a string value to lowercase.	StringValue.**ToLower()**	word.ToLower()	Returns **hello**
IsNumeric	• Takes a string as its argument and returns TRUE if the string contains a number, otherwise it returns false.	IsNumeric(x As String)	IsNumeric(word)	Returns **False**
Length	• Returns the number of characters in a string.	StringValue.**Length**	word.Length	Returns **5**

Substring	• Returns a substring, or a string within a string. • The first format takes an Integer parameter (i.e. **start**) indicating the starting position of the String to be extracted. • The second format takes a second Integer paramter (i.e. **length**) which specifies the number of characters to extract.	StringValue.**Substring** (start As Integer) OR StringValue.**Substring** (start As Integer, length As Integer)	word.Substring(3) word.Substring (1, 3)	Returns **lo** Returns **ell**
IndexOf	• Searches for a character or a string within a string and returns the character position, or index, of the first occurrence of the character. • The first format takes a String parameter which is the value that will be searched. • The second format takes a second Integer parameter, which specifies the starting position within the word to begin the search. • The third format takes a third Integer parameter specifying the number of characters to search.	StringValue.**IndexOf** (x As String) StringValue.**IndexOf** (x As String, start As Integer)	word.IndexOf("e") word.IndexOf ("e", 2)	Returns **1** Returns **-1**
Replace	• Replaces all occurrences of a specified character with another specified character	StringValue.**Replace** (oldChar As String, newChar As String)	word.Replace ("l", "g")	Returns **Heggo**
Insert	• Inserts a string value at a specified index.	StringValue.**Insert**(start As Integer, x As String)	word.Insert (5, "ooo!")	Returns **Helloooo!**
Remove	• Deletes a specified number of characters beginning at a specified position • The first format removes all characters from the starting point specified • The second format takes a second Integer parameter specifying the number of characters to remove.	StringValue.**Remove** (start As Integer) OR StringValue.**Remove**(start As Integer, end As Integer)	word.Remove(2) word.Remove(2, 2)	Returns **he** Returns **heo**
TrimStart	• Returns a copy of a String with all leading white spaces removed.	*StringValue*.TrimStart		
TrimEnd	• Returns a copy of a String with all trailing spaces removed.	*StringValue*.TrimEnd		
Trim	• Returns a copy of a String with all leading and trailing spaces removed.	*StringValue*.Trim		