Waunakee Community High School

Computer Science I

Lesson 04

Part 2

(The PDF version. Video version will be posted when I'm not sick.)

Concepts

- Static
- Off-By-One
- String functions
  - .Length
  - .CompareTo()
  - .Equals()
  - .IndexOf()
  - .Replace()
  - .Substring()
  - .ToLower()
  - .ToUpper()
  - .Trim()
  - .IsNumeric()
- vbCrLf
- immutable

Off-By-One Error

```
intI = 1

Do While intI < 5
     'Code
     intI += 1
Loop
```

I want this loop to run five times. However, it will only run four times. The problem is a logical error: the code will run when intI is 1, 2, 3, and 4, but not when it's 5.

This is a mistake that is made by all programmers pretty consistently, from novices to veterans. It happens so often that it has several names, including OBOE (for "off-by-one-error"), OB1, and even Obi-Wan.

As the real Obi-Wan would say, be mindful of your code. Or he would say that if he was teaching you how to write program code.

Static Variables

```
Private Sub btnFoo_Click(sender As Object, e As EventArgs) Handles bntFoo.Click
        Dim intFoo as Integer, intBar as Integer
        Static intZim as Integer

        intFoo += 1
        intBar += 1
        intZim += 1

        MsgBox(intFoo & intBar & intZim, , "Variable Values")
End Sub
```

Static variables are like local variables, except that they have a lifetime of the program, and not just the Sub in which they're declared. In the code above, intFoo and intBar are initialized to 0, have 1 added to their values, are displayed in the MsgBox, and finally vanish at the end of the Sub. The Static variable intZim is only initialized (to 0) the first time it is declared. It is also incremented and displayed, but its lifetime doesn't end at the End Sub. Instead, it sort of goes into stasis, and the next time btnFoo is clicked, the value is taken out of that stasis and used. So intFoo's and intBar's values will always display as 1, but intZim's value will continue to increment with each click of btnFoo.

Strings

We already know that strings are just an ordered arrangements of characters held in memory and given a certain name (like strFoo). The computer can't really interpret what is in that string; it's just data to VB. As a result, Strings are *immutable,* which means once they're in there, they can't be changed. When you "change" a String variable's value, what's really happening is that the old data is being forgotten and the variable name is being assigned a shiny new value.

This can make the manipulation of Strings a little weird. The good news is that String variables in VB come with a number of built-in functions and properties to help manipulate them. You can access them the same way you would access the properties of control objects on the form, with the dot notation.

Here are a bunch of String functions (and one property) that are useful.

.Length

```
Dim strFoo as String
strFoo = "Mr. Pavao"

lblFoo.Text = strFoo.Length
```

The Length property of a String is the String's length in characters.

In this case, lblFoo will display 9.

.CompareTo(String)

```
Dim strFoo as String, strBar as String
strFoo = "Mr. Pavao"
strBar = "Mr. Kersten"

lblFoo.Text = strFoo.CompareTo(strBar)
```

The CompareTo function looks at the string you're using and the string in the argument and compares them to see which one should come first, then it returns an integer.

When the string doing the function call (strFoo in this case) should come first, the return value will be negative. If the argument should come first, the return value will be positive. If the two Strings are equal, then the return value will be 0.

The comparison uses ASCII values, so all capital letters come before all lower-case letters.

lblFoo will display a positive number. I think it would be a 5, but that's just a guess.

.Equals(String)

```
Dim strFoo as String
strFoo = "Mr. Pavao"

lblFoo.Text = strFoo.Equals(strBar)
```

The Equals function returns a Boolean: True when the Strings are of equal value, and False when they're not.

lblFoo will display "False" in this case.

.IndexOf(String)

```
Dim strFoo as String
strFoo = "Mr. Pavao"

lblFoo.Text = strFoo.IndexOf("a")
lblBar.Text = strFoo.IndexOf("ao")
lblZim.Text = strFoo.IndexOf("X")
```

The IndexOf function returns an integer that describes the position of the first instance of the String in the argument. This position is indexed from zero (all good geeks count from zero). If the argument is not found, the function will return -1.

lblFoo will display 5, lblBar will display 7, and lblZim will display -1.

.Replace(String, String)

```
Dim strFoo as String, strBar as String
strFoo = "Mr. Pavao"
strBar = "Mr. Kersten"

lblFoo.Text = strFoo.Replace("vao","ulson")
lblBar.Text = strBar.Replace("e","opo")
```

The Replace function replaces all instances of the first argument with the second argument, and then returns the resulting String.

IMPORTANT: Strings are immutable, remember? That means the original String isn't changed. Replace returns a new String based on the old one. If you want to change the old string, you have to assign the Replace return value to the original String.

lblFoo will display "Mr. Paulson"
lblBar will display "Mr. Koporstopon"

.Substring(Integer, Integer)

```
Dim strFoo as String
strFoo = "Mr. Pavao"

lblFoo.Text = strFoo.Substring(4,5)
```

The Substring function returns a string that is a part of the original string. The first argument is the position in the String to start (remembering that it's indexed from 0), and the second argument is the length of the substring.

IMPORTANT: Strings are immutable, remember? That means the original String isn't changed. Substring returns a new String based on the old one. If you want to change the old string, you have to assign the Substring return value to the original String.

lblFoo will display "Pavao"

.ToLower()

```
Dim strFoo as String
strFoo = "Mr. Pavao"

lblFoo.Text = strFoo.ToLower()
```

The ToLower function returns the string with all letters in lower case. Yep, that's about it.

IMPORTANT: Strings are immutable, blah, blah, you should know this by now.

lblFoo will display "mr. pavao"

.ToUpper()

```
Dim strFoo as String
strFoo = "Mr. Pavao"

lblFoo.Text = strFoo.ToUpper()
```

Same thing, different case. Yay.

lblFoo will display "MR. PAVAO"

.Trim()

```
Dim strFoo as String
strFoo = "    Mr. Pavao    "

lblFoo.Text = strFoo.Trim()
```

Trim returns a version of the String without leading and trailing whitespace characters.

strFoo will display "Mr. Pavao"

.IsNumeric()

```
Dim strFoo as String, strBar as String
strFoo = "Mr. Pavao"
strBar = "23.6"

lblFoo.Text = strFoo.IsNumeric()
lblBar.Text = strBar.IsNumeric()
```

IsNumeric returns a Boolean: True when the String can be used as a number, False when it cannot. It's great for preventing your program from crashing because the user entered a non-numeric value in a TextBox intended for a number.

lblFoo will display False
lblBar will display True

vbCrLf

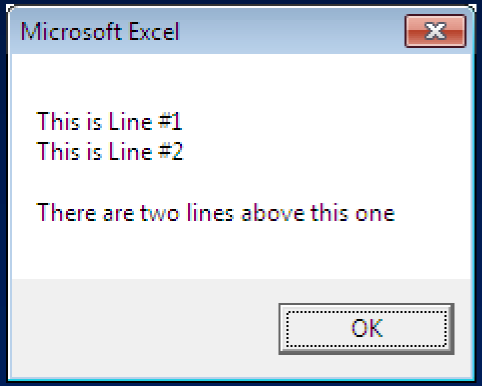
vbCrLf is a built-in constant in VB. It's the character (or actually a combination of two characters) that means "this is the end of a line." It's a line break, an "enter" if you will.

It looks like this.

```
Dim strFoo as String

strFoo = "This is Line #1" & vbCrLf & "This is Line #2"
strFoo = strFoo & vbCrLf & vbCrLf
strFoo = strFoo & "There are two lines above this one"

MsgBox(strFoo, , "Microsoft Excel")
```



```
This is Line #1
This is Line #2

There are two lines above this one
```

That's it for now. I'll be making this into a proper video and adding comprehension questions later. For now, you should be able to get started on the new labs, which are posted on the Web site.

DFTBA!