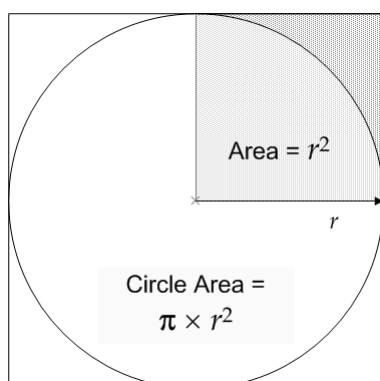


- \* VB .NET Syntax III
- 1. VB Built-in functions (String and Math classes)
- 2. Collections: Array and List (Of T)

## GEOG 4/590: GIS Programming

### Lab 2: Algorithm



- $\text{Pts density} = \#pts / \text{Area}$
- When pts density holds constant, then...
- Given that:  
 $PI = \text{AreaOfCircle}_r / \text{AreaOfSquare}_r$
- We can generate random points in a  $r$  by  $r$  square and see how many of them fall within a radius of  $r$  from a corner of the square...

## OOP - Class Revisit

- Class members
  - Constructors
  - Properties
  - Methods
  - Inheritance (`Inherits BaseClass`)
  - Interfaces

## Object Class

- The base class of all classes in .NET

### Methods

	Name	Description
◆ X	<code>Equals(Object)</code>	Determines whether the specified Object is equal to the current Object.
◆ S X	<code>Equals(Object, Object)</code>	Determines whether the specified object instances are considered equal.
◆ X	<code>Finalize</code>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.
◆ X	<code>GetHashCode</code>	Serves as a hash function for a particular type.
◆ X	<code>GetType</code>	Gets the Type of the current instance.
◆ X	<code>MemberwiseClone</code>	Creates a shallow copy of the current Object.
◆ S X	<code>ReferenceEquals</code>	Determines whether the specified Object instances are the same instance.
◆ X	<code>ToString</code>	Returns a string that represents the current object.

## Using Methods of VB Objects

- objVariable vs. [object]

```
Option Strict On

Public Class Form1
    Private Sub Button2_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button2.Click
        Dim obj1 As New Object
        Dim obj2 As New Object
        Debug.Print(CStr(obj1.Equals(obj2)))
        Debug.Print(CStr([Object].Equals(obj1, obj2)))
        obj2 = obj1
        Debug.Print(CStr(obj1.Equals(obj2)))
        Debug.Print(CStr([Object].Equals(obj1, obj2)))
    End Sub
End Class
```

## String and Char

- FRANK is a string – a collection of chars
- "F", "R", "A", "N", "K" are chars

String contents	F	R	A	N	K	nul	Null-terminator
ASCII code in hex	46	52	41	4E	4B	00	
Index of string	0	1	2	3	4	OR	Index out of range (RTE)

## String Class and String

- Inheritance: System.Object
- Namespace: System.String

Imports System.String

- Properties

```
Public Sub StringProperties()
    Dim str1 As String = "Test"
    For ctr As Integer = 0 To str1.Length - 1
        Console.WriteLine("{0} {1} ", str1(ctr), str1.Chars(ctr))
    Next
End Sub
```

## String Method - IndexOf

```
Public Sub StringMethods()
    Dim str1 As String = "1234567890"
    Dim str2 As String = String.Concat(str1, "abcde", str1) 'concatenation

    Dim indexfirst, indexlast As Integer

    'find the index of first appearance of a char
    indexfirst = str2.IndexOf("5")
    'find the index of the last appearance of a char
    indexlast = str2.LastIndexOf("5")

    Console.WriteLine _
        ("5 first appears at index {0} and last appears at index {1}", _
        indexfirst, indexlast)
End Sub
```

## String Method - Compare

```
Public Sub StringCompare()  
    Dim lastName As String = "brown"  
  
    MsgBox(lastName.ToUpper)      'return "BROWN"  
    MsgBox(String.Compare(lastName, "WHITE")) 'return -1  
    MsgBox(String.Compare(lastName.ToUpper, "BROWN")) 'return 0  
    MsgBox(String.Compare(lastName, "apricot", True)) 'return 1  
End Sub
```

## String Method - IsNullOrEmpty

```
Public Sub StringIsNullOrEmpty()  
    Dim str1 As String = "abc"  
    Dim str2 As String = " "  
    Dim str3 As String = ""  
    Dim str4 As String = Nothing  
  
    Console.WriteLine("str1 is null or empty: {0}. Its length is {1}", _  
        String.IsNullOrEmpty(str1), str1.Length)  
    Console.WriteLine("str2 is null or empty: {0}. Its length is {1}", _  
        String.IsNullOrEmpty(str2), str2.Length)  
    Console.WriteLine("str3 is null or empty: {0}. Its length is {1}", _  
        String.IsNullOrEmpty(str3), str3.Length)  
    Console.WriteLine("str4 is null or empty: {0}. Its length is {1}", _  
        String.IsNullOrEmpty(str4), str4.Length)  
End Sub
```

## String Method – Insert, Trim

```
Public Sub StringInsert()
    Dim one As String = " one "
    Dim two As String = " two "
    Dim three As String = " three "
    Dim str123 As String = String.Concat(one, three)

    Console.WriteLine("Original string: {0}", str123)

    'insert two into str123
    str123 = str123.Insert(str123.IndexOf(three), two.Trim)

    Console.WriteLine("Final string: {0}", str123)
End Sub
```

## String Method – Split, Substring PadLeft, PadRight

```
Public Sub StringPadSplit()
    Dim str1 As String = "abc"
    Dim str2 As String = "abcdefg"
    Dim str3 As String = "abcdefghijklmn"

    'padding strings
    Console.WriteLine(" PadLeft(10): {0}, {1}, {2}", _
        str1.PadLeft(10), str2.PadLeft(10), str3.PadLeft(10))
    Console.WriteLine(" PadRight(10): {0}, {1}, {2}", _
        str1.PadRight(10), str2.PadRight(10), str3.PadRight(10))

    'split strings
    Dim str123 As String = str1 & "," & str2 & "," & str3
    Console.WriteLine("Before split: {0}", str123)

    'Dim strArray() As String
    'strArray = str123.Split(",")
    For Each i As String In str123.Split(",")
        Console.WriteLine(i)
    Next

    'substring
    Console.WriteLine("Substring after {0},{0}: " & _
        str123.Substring(str123.IndexOf(",") + 1), Chr(34))
End Sub
```

## String Method – Replace, Remove

```
Dim correctString As String = errString.Replace("docment", "document")
```

```
Public Sub StringRemove()
    Dim name As String = "Michelle Violet Banks"
    Dim foundS1 As Integer = name.IndexOf(" ")
    Dim foundS2 As Integer = name.IndexOf(" ", foundS1 + 1)

    ' remove the middle name, identified by finding
    ' the spaces in the middle of the name...
    If foundS1 >= 0 And foundS1 <> foundS2 Then
        name = name.Remove(foundS1 + 1, foundS2 - foundS1)
        Console.WriteLine(name)
    End If
End Sub
```

## Math Class

- Inheritance: System.Object
- Namespace: System.Math
  - Imports System.Math
  - Math.Sqrt() 'all methods are static members
- Properties – none
- Fields
  - $\pi$  (PI) – a double constant (3.14159265358979323846)
    - Math.PI
  - e (E) – natural logarithmic base (2.7182818284590452354)
    - Math.E

## Math Method – Abs, Pow, Sqrt

```
Public Sub MathBasic()
    Dim outputValue As Double
    Dim inputValue As Double

    inputValue = 2
    outputValue = _
        Math.Sqrt(Math.Abs(Math.Pow(inputValue, 3.2) - inputValue ^ 5))

    MsgBox(outputValue)
End Sub
```

## Math Method – Truncate, Round, Floor, Ceiling

Methods	Descriptions	4.85	3.14	-1.25	-3.72
Ceiling	Returns the smallest integer larger than or equal to a number	5	4	-1	-3
Floor	Returns the largest integer less than or equal to a number	4	3	-2	-4
Round	Rounds a number to the nearest integer	5	3	-1	-4
Truncate	Returns the integer part of a number	4	3	-1	-3
CLng, CInt	Same as Round	5	3	-1	-4
Int	Same as Floor	4	3	-2	-4

You can also round a number to a decimal place you specified in the method.  
`outValue = Math.Round(inValue, 2)`



## Side Notes for Geeks...

1. Math.Ceiling(number)
2. Math.Floor(number)
3. Math.Round(number + 0.5)
4. Math.Round(number - 0.5)

## Math Method – DivRem Mod (VB operator)

- Quotient and remainder in integer division

```
Dim quotient As Long = _  
Math.DivRem(dividend, divisor, remainder)  
  
quotient = Math.Truncate(dividend / divisor)  
remainder = dividend Mod divisor
```

## Math Method - Trigonometry

- Angle values are in radian
- Convert degree to radian first!

$$1 \text{ degree} = \pi / 180$$

```
Dim angle As Double = degrees * Math.PI / 180.0
Math.Sin(angle)
```

## Array

- Array declaration

```
Dim players(9) As String `static array
players(0) = "Adam"
players(1) = "Brad"
...
players(8) = "Henry"
players(9) = "Ian"
Console.WriteLine(players.Count)
```

```
Dim nextplayer As String = players(5)
Dim scores() As Integer = {1, 4, 5, 8}
```

## Passing Arrays As Parameters








```
Public Sub ArrayAsParameter()
    Dim players(4) As String
    players(0) = "A"
    players(1) = "B"
    players(2) = "C"
    players(3) = "D"
    players(4) = "E"
    PrintArray(players)
End Sub
```

---

```
Private Sub PrintArray(ByVal strArray() As String)
    For Each i As String In strArray
        Console.WriteLine(i)
    Next
End Sub
```

## Array Class

- Inheritance: System.Object
- Namespace: System.Array
- Properties:

	Name	Description
 X	IsFixedSize	Gets a value indicating whether the Array has a fixed size.
 X	IsReadOnly	Gets a value indicating whether the Array is read-only.
 X	IsSynchronized	Gets a value indicating whether access to the Array is synchronized (thread safe).
 X	Length	Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
	LongLength	Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
 X	Rank	Gets the rank (number of dimensions) of the Array.
 X	SyncRoot	Gets an object that can be used to synchronize access to the Array.

## Array Method - IndexOf

```
Public Sub ArrayIndexOf()
    Dim dinosaurs() As String = _
        {"Tyrannosaurus", _
        "Amargasaurus", _
        "Mamenchisaurus", _
        "Brachiosaurus", _
        "Deinonychus", _
        "Tyrannosaurus", _
        "Compsognathus"}

    Console.WriteLine(vbLf & _
        "Array.IndexOf(dinosaurs, ""Tyrannosaurus"): {0}", _
        Array.IndexOf(dinosaurs, "Tyrannosaurus"))

    Console.WriteLine(vbLf & _
        "Array.IndexOf(dinosaurs, ""Tyrannosaurus"", 3): {0}", _
        Array.IndexOf(dinosaurs, "Tyrannosaurus", 3))

    Console.WriteLine(vbLf & _
        "Array.IndexOf(dinosaurs, ""Tyrannosaurus"", 2, 2): {0}", _
        Array.IndexOf(dinosaurs, "Tyrannosaurus", 2, 2))
End Sub
```

## Array Method – Resize, Clear

```
Public Sub ArrayResize()
    Dim myArr As String() = _
        {"1", "2", "3", "4", "5", "6", "7", "8", "9"}
    Dim i As Integer

    ' Resize the array to a bigger size (five elements larger).
    Array.Resize(myArr, myArr.Length + 5)
    For i = 0 To myArr.Length - 1
        Console.WriteLine("[{0}]: {1}", i, myArr(i))
    Next

    ' Resize the array to a smaller size (four elements).
    Array.Resize(myArr, 4)
    For i = 0 To myArr.Length - 1
        Console.WriteLine("[{0}]: {1}", i, myArr(i))
    Next

    ' Reset the elements to empty
    Array.Clear(myArr, 0, myArr.Length)
    For i = 0 To myArr.Length - 1
        Console.WriteLine("[{0}]: {1}", i, myArr(i))
    Next
End Sub 'Main
```

## Dynamic Array – ReDim [Preserve]

```
Public Sub ArrayDynamic()
    Dim myArr(8) As String 'static array
    Dim i As Integer

    For i = 0 To myArr.Length - 1
        myArr(i) = i + 1
    Next

    ' Resize the array to a bigger size (five elements larger).
    ReDim myArr(myArr.Length + 5)
    'Array.Resize(myArr, myArr.Length + 5)
    For i = 0 To myArr.Length - 1
        Console.WriteLine("{0}: {1}", i, myArr(i))
    Next

    For i = 0 To myArr.Length - 1
        myArr(i) = i + 1
    Next

    ' Resize the array to a smaller size (four elements).
    ReDim Preserve myArr(4)
    For i = 0 To myArr.Length - 1
        Console.WriteLine("{0}: {1}", i, myArr(i))
    Next
End Sub 'Main
```

## Array Method – Sort, Reverse

```
Public Sub ArraySort()
    Dim dinosaurs() As String = _
        {"Tyrannosaurus", _
        "Amargasaurus", _
        "Mamenchisaurus", _
        "Brachiosaurus", _
        "Deinonychus", _
        "Tyrannosaurus", _
        "Compsognathus"}

    Array.Sort(dinosaurs) 'sort ascendingly
    For Each i As String In dinosaurs
        Console.WriteLine(i)
    Next

    Array.Reverse(dinosaurs) 'sort descendingly
    For Each i As String In dinosaurs
        Console.WriteLine(i)
    Next
End Sub
```

## Array Method – Paired Sort

```
Public Sub ArrayPairedSort()
    Dim players() As String = _
        {"Sid", "Adam", "Cliff", "Zack", "David", "Gil", "Mica"}
    Dim battingorder() As Integer = _
        {4, 7, 3, 2, 6, 1, 5}

    For i As Integer = 0 To players.Length - 1
        Console.WriteLine(players(i) & " batting number is " & battingorder(i))
    Next

    Array.Sort(battingorder, players) 'paired sort using battingorder as a key
    For i As Integer = 0 To players.Length - 1
        Console.WriteLine(players(i) & " batting number is " & battingorder(i))
    Next
End Sub
```

## Multi-dimensional Array

```
Public Sub ArrayMultiD()
    Dim diagonal(,) As Single = New Single(,) {{1, 0}, {0, 1}}
    'list the total number of elements
    Console.WriteLine(diagonal.Length)

    Dim matrix(5, 9) As Double
    Console.WriteLine(matrix.Length)

    Dim maxDim0 As Integer = UBound(matrix, 1)
    'Dim maxDim0 As Integer = matrix.GetLength(0)

    Dim maxDim1 As Integer = UBound(matrix, 2)
    'Dim maxDim1 As Integer = matrix.GetLength(1)

    For i As Integer = 0 To maxDim0
        For j As Integer = 0 To maxDim1
            matrix(i, j) = (i * 10) + j
        Next j
    Next i
End Sub
```

0	1	2	3	4	5	6	7	8	9
10	11	12	13	...					
									59

## What is Enumeration?

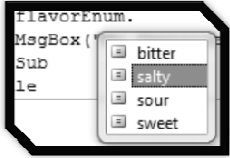
- Enumerations offer an easy way to work with sets of related constants and to associate constant values with names
- Member name [= initializer]

```

Module Module2
    Public Enum flavorEnum
        salty
        sweet
        sour
        bitter
    End Enum

    Public Sub TestMethod()
        MsgBox("My favorite is " & flavorEnum.salty)
    End Sub
End Module

```



## Enumeration

- Enumerations make for clearer and more readable code, particularly when meaningful names are used.
- Benefits :
  - Reduces errors caused by transposing or mistyping numbers
  - Makes it easy to change values in the future
  - Makes code easier to read, which means it is less likely that errors will creep into it
  - Ensures forward compatibility

## Structures

```
Public Structure SystemInfo
    Public CPU As String
    Public Memory As Long
    Public PurchaseDate As Date
End Structure

Public Class Class1
    Public Function MyFunction
        Dim MySystem, YourSystem As SystemInfo

        MySystem.CPU = "486"
        Dim TooOld As Boolean
        If YourSystem.PurchaseDate < #1/1/1992# Then TooOld =
            True

        YourSystem = MySystem
    End Function
End Class
```

## Naming Convention

- VB .NET is NOT case sensitive, however...
- Names
  - A list of identifiers concatenated to form a name
- Pascal case
  - ErrorLevel
- Camel case
  - totalNetworkDistance
- Uppercase
  - System.IO



## Naming Convention - Principles

- Be consistent!
- Don't use "reserved" terms (e.g., `Double`, `False`)
- Use Pascal casing for namespaces, classes, members, methods, and constants  

```
Const MapUnits As String = "meters"
```
- Use Camel casing for variables and parameters  

```
Dim isVisible As Boolean
```
- Do not use underscores, hyphens, or any other nonalphanumeric characters

## Data Structures

- Array
- Arraylist (queue and stack)
- Hashtable
- Binary (search) tree
- Graph