What do we need to learn in order to write computer programs?

- Fundamental programming constructs:
  - Variables,
  - Arithmetic operators,
  - Input and output
  - Conditionals,
  - Loops,
  - Procedures and functions,
  - Arrays (Multi-Dimensional Arrays),
  - Structures, **classes and objects**, 
  - Files
What is Object-Oriented Paradigm?

- **MODELING THE REAL WORLD**

  - Humans **natural ability to classify, generalize and create abstractions** in order to model the world.

  - **View the world as a series of entities** and the interaction among them.

  - Problems are solved through the interaction among a number of **autonomous** and **cooperative** objects.
What is Object-Oriented Paradigm?

- View the **Code and Data** as an **atomic and non-decomposable unit**.

- Traditional programming languages deliver modularization by the use of procedures and functions.
What is Object-Oriented Paradigm?

- An object is defined as a set of **built-in and/or user-defined data types along with a set of operations** which manipulate them.

- **Object = Data + Code**

- The functions which surround the data structure provide a **robust interface to the clients** of that data.

- **Simplifies the debugging, maintenance and testing.**

- The applications or clients which use object will not be effected by the possible changes in the implementation of the object, as long as the interface or behavior of the object remains constant.
The 3 Pillars of Object-Oriented Paradigm

1. Encapsulation
2. Inheritance
3. Polymorphism
Encapsulation

- Encapsulation, **makes large projects intellectually manageable**.

- Encapsulation is achieved by **tightly coupling the data structure and its related methods (functions)** hence viewing them as an atomic unit.

- Access to the data is **only** possible through the **public interface** provided. Encapsulation hides the details of the object from the client.
Inheritance

- One of the most important aspect of object oriented systems.

- A new class can inherit **reuse the behavior and structure** of previously defined classes.

- The Sub-Classes (Derived Classes) can **extend or add more functionality** (i.e. methods and instance variables) to the base class.

- Inheritance is a **natural tool to express relationships** such as classification, specialization, generalization, evolution and approximation.
Polymorphism comes from the Greek: "having many shapes".
Polymorphism

- Two methods for achieving Polymorphism:
  - Overloading
  - Overriding

- More later.....
Example of Encapsulation:

Public Class CAR
    ' Private Data
    Private color As Integer
    Private year As Integer
    Private make As String
    Private model As String

    '-------------------------------
    ' Public Methods
    Public Sub PrintColor()
        Console.WriteLine("The Color = {0}", color)
    End Sub
    '-------------------------------
    Public Sub SetColor(ByVal clr As Integer)
        color = clr
    End Sub
End Class
Example of Encapsulation:

Class CAR

' Private Data
Private color As String
Private year As Integer
Private make As String
Private model As String

' Public Methods
Public Sub SetColor(ByVal clr As String)
    color = clr
End Sub

Public Sub PrintColor()
    Console.WriteLine("The Color = {0}", color)
End Sub

End Class
Example of Encapsulation:

Module Module1

Sub Main()

Dim MyCar As CAR
MyCar = New CAR
' MyCar is a Reference Variable
' The NEW operator will actually allocate the memory
' needed for the car object.

MyCar.SetColor("GREEN")

MyCar.PrintColor()

Console.WriteLine(MyCar.color)  ' Note that this is a error, since we don't have
' direct access to the Color attribute.

End Sub

End Module
Example of Inheritance:

' The SUPER_CAR class Inherits from the CAR class
' and add extends the class by adding a new attribute
' and two new methods.

Class SUPER_CAR
    Inherits CAR
        ' Inherits from the CAR class

    Public SuperAttribute As String

    '-------------------------------
    Public Function GetSuperAttribute() As String
        Return SuperAttribute
    End Function

    '-------------------------------
    Public Sub SetSuperAttribute(ByVal SuperAttr As String)
        SuperAttribute = SuperAttr
    End Sub

End Class
Example of Inheritance:

Module Module1

Sub Main()

    Dim MyCar As CAR
    MyCar = New CAR
    MyCar.SetColor("GREEN")
    MyCar.PrintColor()

    Dim BatMobil As New SUPER_CAR
    BatMobil.SetColor("BLACK")
    BatMobil.SetSuperAttribute("Goes Real Fast!!")
    BatMobil.PrintColor()

    Console.WriteLine(BatMobil.SuperAttribute) 'Note that we can access SuperAttribute because it is public.

End Sub

End Module
Polymorphism:

- Polymorphism allows the programmer to use the same method name to perform a “given generic task” on a number of different data types.

- It also allows the programmer to use the same method name to perform different functions.

- This is known as overloading.
Example of Polymorphism:

- Imagine in our Car class, we have two methods for setting the color:

```csharp
Class CAR

    ' Private Data
    Private color As String
    Private year As Integer
    Private make As String
    Private model As String

    ' Public Methods
    Public Sub SetColor(ByVal clr As String)
        color = clr
    End Sub

    Public Sub SetColor()
        color = "YELLOW"
    End Sub

    Public Sub PrintColor()
        Console.WriteLine("The Color = {0}", color)
    End Sub

End Class
```

The first SetColor() method takes an argument and sets the color or the car to that argument.
Example:
```
SetColor("GREEN")
```

The second SetColor() method takes NO arguments and sets the color or the car to a default value “YELLOW”.
Example:
```
SetColor()
```