Part 1 - C# Tutorial - Introduction

In this session  
1. We will learn the basic structure of a c# program. The program we used in this video is shown below.  
  
// Namespace Declaration  
using System;  
  
  
class Pragim  
{  
    public static void Main()  
    {  
        // Write to console  
        Console.WriteLine ("Welcome to PRAGIM Technologies!");   
    }  
}  
  
  
  
2. Understand the purpose of **using System** declaration - The namespace declaration,**using System**, indicates that you are using the System namespace. If  you omit the **using System**, declaration, then you have to use the fully qualified name of the **Console**class. A namespace is used to organize your code and is collection of classes, interfaces, structs, enums and delegates. We will discuss about namespaces in detail in a later session.  
  
3. Purpose of**Main()** method - Main method is the entry point into your application.

Part 2 - C# Tutorial - Reading and writing to a console

**Suggested Videos**

[Part 1 - Introduction to C#](http://csharp-video-tutorials.blogspot.com/2012/06/part-1-c-tutorial-introduction.html)

**In this video, we will discuss**  
**1.** Reading from the console  
**2.** Writing to the console  
**3.** Two ways to write to console  
     **a)** Concatenation  
     **b)** Place holder syntax – Most preferred   
  
   
  
**Code samples used in the demo**  
using System;  
class Program  
{  
    static void Main()  
    {  
        // Prompt the user for his name  
        Console.WriteLine("Please enter your name");  
        // Read the name from console  
        string UserName = Console.ReadLine();  
        // Concatenate name with hello word and print  
        Console.WriteLine("Hello " + UserName);  
  
        //Placeholder syntax to print name with hello word   
        //Console.WriteLine("Hello {0}", UserName);  
    }  
}  
  
Please note that C# is **case sensitive language**.

Part 3 - C# Tutorial - Built-in types

**Suggested Videos**  
[Part 1 - Introduction to C#](http://csharp-video-tutorials.blogspot.com/2012/06/part-1-c-tutorial-introduction.html)  
[Part 2 - Reading and writing to console](http://csharp-video-tutorials.blogspot.com/2012/06/part-2-c-tutorial-reading-and-writing.html)   
  
In this video, we will discuss the different **built-in types** that are available in c#.   
  
**Built-in types in C#**  
**1.** Boolean type – Only true or false   
**2.** Integral Types - sbyte, byte, short, ushort, int, uint, long, ulong, char  
**3.** Floating Types – float and double  
**4.** Decimal Types   
**5.** String Type    
  
**Escape Sequences in C#**  
<http://msdn.microsoft.com/en-us/library/h21280bw.aspx>  
  
Verbatim Literal is a string with an @ symbol prefix, as in @“Hello".   
  
Verbatim literals make escape sequences translate as normal printable characters to enhance readability.   
  
**Practical Example:**  
**Without Verbatim Literal :** “C:\\Pragim\\DotNet\\Training\\Csharp” – Less Readable  
**With Verbatim Literal :** @“C:\Pragim\DotNet\Training\Csharp” – Better Readable

Part 4 - C# Tutorial - String type

**Suggested Videos**  
[Part 1 - Introduction to C#](http://csharp-video-tutorials.blogspot.com/2012/06/part-1-c-tutorial-introduction.html)  
[Part 2 - Reading and writing to console](http://csharp-video-tutorials.blogspot.com/2012/06/part-2-c-tutorial-reading-and-writing.html)  
[Part 3 - Built-in types](http://csharp-video-tutorials.blogspot.com/2012/06/part-3-c-tutorial-built-in-types.html)

Part 5 - C# Tutorial - Common Operators

**Suggested Videos**  
[Part 2 - Reading and writing to console](http://csharp-video-tutorials.blogspot.com/2012/06/part-2-c-tutorial-reading-and-writing.html)  
[Part 3 - Built-in types](http://csharp-video-tutorials.blogspot.com/2012/06/part-3-c-tutorial-built-in-types.html)  
[Part 4 - String type](http://csharp-video-tutorials.blogspot.com/2012/06/part-4-c-tutorial-string-type.html)   
  
In this video, we will discuss the **common operators that are available in c# programming language**.   
  
Assignment Operator **=**  
Arithmetic Operators like **+,-,\*,/,%**  
Comparison Operators like **==, !=,>, >=, <, <=**  
Conditional Operators like **&&, ||**  
Ternary Operator **?:**  
Null Coalescing Operator **??**  
  
**Program without ternary Operator**  
using System;  
class Program  
{  
    static void Main()  
    {  
        int Number = 10;  
  
        bool IsNumber10;  
  
        if (Number == 10)  
        {  
            IsNumber10 = true;  
        }  
        else  
        {  
            IsNumber10 = false;  
        }  
  
        Console.WriteLine("i == 10 is {0}", IsNumber10);  
    }  
}  
**Same program with ternary Operator**  
using System;  
class Program  
{  
    static void Main()  
    {  
        int Number = 10;  
  
        bool IsNumber10 = Number == 10 ? true : false;  
          
        Console.WriteLine("i == 10 is {0}", IsNumber10);  
    }  
}

Part 6 - C# Tutorial - Nullable Types

**Suggested Videos**  
[Part 3 - Built-in types](http://csharp-video-tutorials.blogspot.com/2012/06/part-3-c-tutorial-built-in-types.html)  
[Part 4 - String type](http://csharp-video-tutorials.blogspot.com/2012/06/part-4-c-tutorial-string-type.html)   
[Part 5 - Operators](http://csharp-video-tutorials.blogspot.com/2012/06/part-5-c-tutorial-common-operators.html)  
  
**In this video, we will discuss**  
**1.** Nullable types in C#  
**2.** Null Coalescing Operator ??    
  
**In C# types are divided into 2 broad categories.**  
**Value Types**  - int, float, double, structs, enums etc  
**Reference Types** – Interface, Class, delegates, arrays etc  
  
**By default value types are non nullable. To make them nullable use ?**  
int i = 0 (i is non nullable, so "i" cannot be set to null, i = null will generate compiler error)  
int? j = 0 (j is nullable int, so j=null is legal)  
  
**Nullable types bridge the differences between C# types and Database types**  
  
**Program without using NULL coalescing operator**  
using System;  
class Program  
{  
    static void Main()  
    {  
        int AvailableTickets;  
        int? TicketsOnSale = null;  
  
        if (TicketsOnSale == null)  
        {  
            AvailableTickets = 0;  
        }  
        else  
        {  
            AvailableTickets = (int)TicketsOnSale;  
        }  
  
        Console.WriteLine("Available Tickets={0}", AvailableTickets);  
    }  
}  
  
**The above program is re-written using NULL coalescing operator**  
using System;  
class Program  
{  
    static void Main()  
    {  
        int AvailableTickets;  
        int? TicketsOnSale = null;  
  
        //Using null coalesce operator ??  
        AvailableTickets = TicketsOnSale ?? 0;  
  
        Console.WriteLine("Available Tickets={0}", AvailableTickets);  
    }  
}

Part 7 - C# Tutorial - Datatype conversions

**Suggested Videos**  
[Part 4 - String type](http://csharp-video-tutorials.blogspot.com/2012/06/part-4-c-tutorial-string-type.html)  
[Part 5 - Operators](http://csharp-video-tutorials.blogspot.com/2012/06/part-5-c-tutorial-common-operators.html)   
[Part 6 - Nullable Types](http://csharp-video-tutorials.blogspot.com/2012/06/part-6-c-tutorial-nullable-types.html)   
  
**In this video, we will discuss**  
**1.** Implicit conversions  
**2.** Explicit Conversions  
**3.** Difference between Parse() and TryParse()   
  
**Implicit conversion is done by the compiler:**  
**1.** When there is no loss of information if the conversion is done  
**2.** If there is no possibility of throwing exceptions during the conversion

**Example:**Converting an **int** to a **float** will not loose any data and no exception will be thrown, hence an implicit conversion can be done.   
  
Whereas when converting a **float** to an **int**, we loose the fractional part and also a possibility of overflow exception. Hence, in this case an explicit conversion is required. For explicit conversion we can use cast operator or the convert class in c#.  
  
**Implicit Conversion Example**  
using System;  
class Program  
{  
    public static void Main()  
    {  
        int i = 100;  
  
        // float is bigger datatype than int. So, no loss of  
        // data and exceptions. Hence implicit conversion  
        float f = i;  
  
        Console.WriteLine(f);  
    }  
}  
  
**Explicit Conversion Example**  
using System;  
class Program  
{  
    public static void Main()  
    {  
        float f = 100.25F;  
  
        // Cannot implicitly convert float to int.  
        // Fractional part will be lost. Float is a  
        // bigger datatype than int, so there is  
        // also a possiblity of overflow exception  
        // int i = f;  
  
        // Use explicit conversion using cast () operator  
        int i = (int)f;  
  
        // OR use Convert class  
        // int i = Convert.ToInt32(f);  
  
        Console.WriteLine(i);  
    }  
}  
  
**Difference between Parse and TryParse**  
**1.** If the number is in a string format you have 2 options - Parse() and TryParse()   
**2.** Parse() method throws an exception if it cannot parse the value, whereas TryParse() returns a bool indicating whether it succeeded or failed.  
**3.** Use Parse() if you are sure the value will be valid, otherwise use TryParse()

Part 8 - C# Tutorial - Arrays in C#

**Suggested Videos**  
[Part 5 - Operators](http://csharp-video-tutorials.blogspot.com/2012/06/part-5-c-tutorial-common-operators.html)   
[Part 6 - Nullable Types](http://csharp-video-tutorials.blogspot.com/2012/06/part-6-c-tutorial-nullable-types.html)   
[Part 7 - Datatype Conversions](http://csharp-video-tutorials.blogspot.com/2012/06/part-7-c-tutorial-datatype-conversions.html)  
  
**In this video, we will discuss**  
**1.** Arrays  
**2.** Advantages and dis-advantages of arrays   
  
**An array is a collection of similar data types.**  
  
using System;  
class Program  
{  
    public static void Main()  
    {  
        // Initialize and assign values in different lines  
        int[] EvenNumbers = new int[3];  
        EvenNumbers[0] = 0;  
        EvenNumbers[1] = 2;  
        EvenNumbers[2] = 4;  
  
        // Initialize and assign values in the same line  
        int[] OddNumbers = { 1, 3, 5};  
  
        Console.WriteLine("Printing EVEN Numbers");  
  
        // Retrieve and print even numbers from the array  
        for (int i = 0; i < EvenNumbers.Length; i++)  
        {  
            Console.WriteLine(EvenNumbers[i]);  
        }  
  
        Console.WriteLine("Printing ODD Numbers");  
  
        // Retrieve and print odd numbers from the array  
        for (int i = 0; i < OddNumbers.Length; i++)  
        {  
            Console.WriteLine(OddNumbers[i]);  
        }  
    }  
}  
  
   
  
**Advantages:** Arrays are strongly typed.  
  
**Disadvantages:** Arrays cannot grow in size once initialized. Have to rely on integral indices to store or retrieve items from the array.

Part 9 - C# Tutorial - Comments

**Suggested Videos**  
[Part 6 - Nullable Types](http://csharp-video-tutorials.blogspot.com/2012/06/part-6-c-tutorial-nullable-types.html)  
[Part 7 - Datatype Conversions](http://csharp-video-tutorials.blogspot.com/2012/06/part-7-c-tutorial-datatype-conversions.html)   
[Part 8 - Arrays](http://csharp-video-tutorials.blogspot.com/2012/06/part-8-c-tutorial-arrays-in-c.html)   
  
**In this video, we will discuss**  
**1.** Single line comments  
**2.** Multi line comments   
**3.** Introduction to XML documentation comments    
  
Single line Comments                       -   //  
Multi line Comments                    -   /\*  \*/  
XML Documentation Comments      -   ///  
  
Comments are used to document what the program does and what specific blocks or lines of code do. C# compiler ignores comments.  
  
**To Comment and Uncomment, there are 2 ways**  
**1.** Use the designer  
**2.** Keyboard Shortcut: Ctrl+K, Ctrl+C and Ctrl+K, Ctrl+U  
  
**Note:** Don't try to comment every line of code. Use comments only for blocks or lines of code that are difficult to understand

### Part 10 - C# Tutorial - If statement