C# Notes

- **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

- **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);
    }
}
```

Datatype conversions

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.

Where as 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#.

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 cast operator and Convert class:

Whenever if there is error converting one data type to another, using simply cast operator it does not throws any exception but where as convert class throws an exception.

Example:

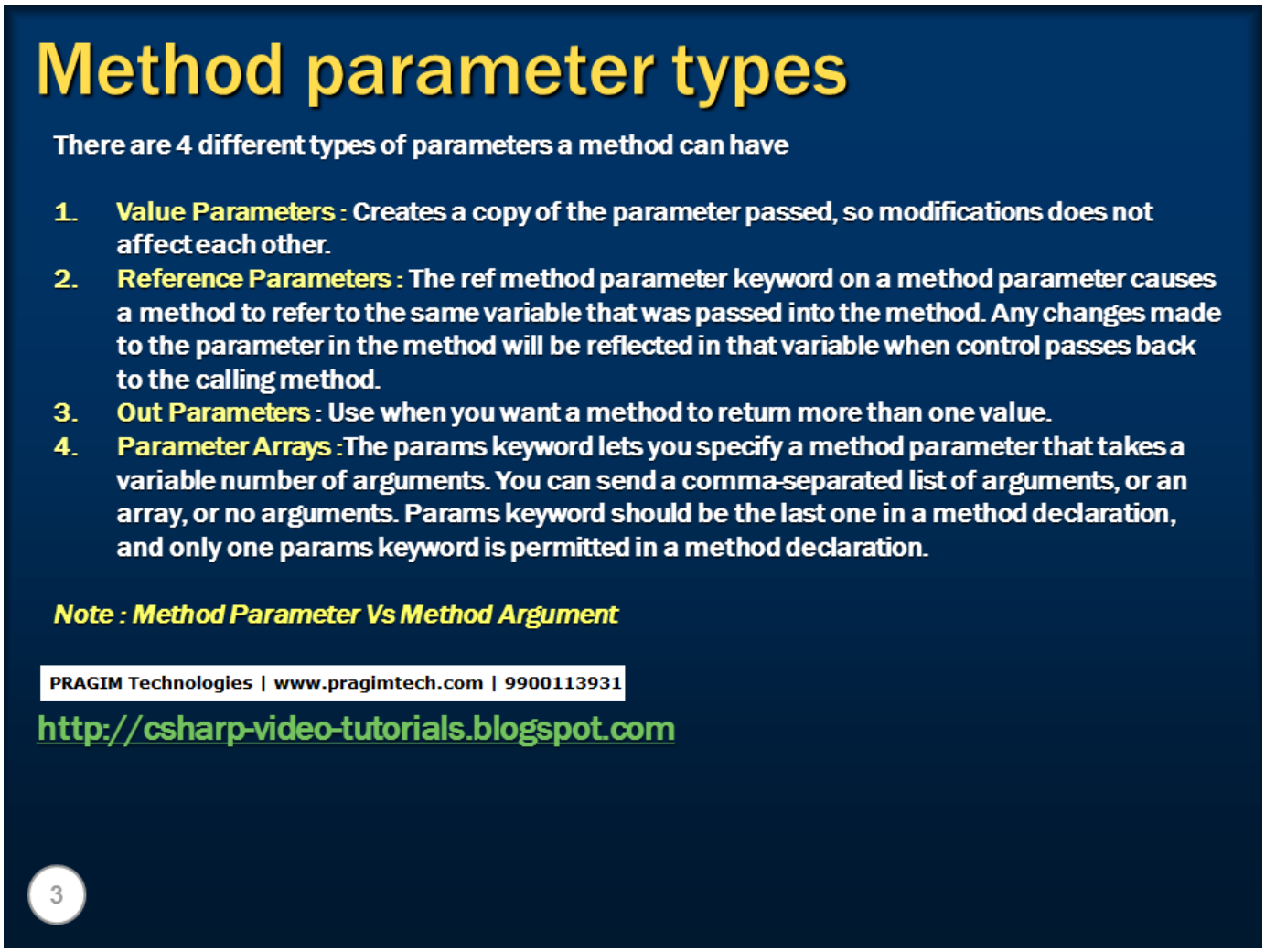
```
using System;


public class HelloWorld
{
    public static void Main(string[] args)
    {
        float f = 1233546789654893.25f;
        //Tryig to convert using cast doesn't throws exception
        //It just displays lowest int value. i.e -2147483648
        int i = (int)f;
        //Using convert class gives out of flow exception
        int j = Convert.ToInt32(f);
    }
}
```

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.

```
using System;
public class HelloWorld
{
    public static void Main(string[] args)
    {
        string demo = "100ff";
            //Using parse methos it throws error if string to be converted in not  a proper number
            int num = int.Parse(demo);
            //If we use Try Parse it returns true or false value
```

```
            int result = 0;
            int.TryParse(demo,out result);
        }
    }
```



Pass By Ref:

```
    public static void Main(string[] args)
    {
        //Pass by ref example
        int a = 10;
        printByRef(ref a);
        Console.WriteLine(a);
    }
    public static void printByRef(ref int a){
        Console.WriteLine(a);
```

```
        //This will increment a by 1
        ++a;
    }
```

Out Parameter :

```
        static void Main(string[] args)
        {
            int sum = 0;
            int product = 0;
            sumandprod(3, 5, out sum, out product);
            Console.Write("Sum of number is {0} && Product is {1}",sum,product);
        }

        static void sumandprod(int fn, int sn, out int sum, out int prod)
        {
            sum = fn + sn;
            prod = fn * sn;
        }
```

Params:

```
        static void Main(string[] args)
        {
            paramDemo();
            paramDemo(3, 5, 6, 7);
            int[] a = { 20, 40 };
            paramDemo(a);
        }

        static void paramDemo(params int[] a)
        {
            foreach (int k in a)
                Console.WriteLine(k);
        }
```