7. Functions and Exception Handling in C#.Net

C# Code that must be executed many times is a good candidate for functions. For Example, if you have to perform some operation that required you to cut and paste the same code over and over again can get pretty tedious. Also, C# function can provide a centralized location for common information. This way if you have to change common code you do not have to search and replace text in your code, you can just change the code in the function.

```
[Access modifier] <return type> <Name of the Function>([Parameters])
{
        <Body of the function>
}
e.g.
Public static int Add(int No1,int No2)
{No1+No2;}
```

7.1 Return Values in a C# Method

In a C# function you may want to return the result of an operation. You can return the value of a variable in a function by using the return keyword.

If u place only single statement without return type it will take u out of code block. void is also return type of null.

Demo 7.1: How to return values from functions.

7.2 C# Call by Reference:

In the previous sections we have been calling functions by value. Now we will learn how to call by reference. Use call by reference any chance you get because it has tremendous performance benefits. The compiler does not have to spend the time to create a temporary variable, it uses the same variable in the calling statement. So, any changes that are made to the variable are in effect through out the whole program.

•When you call by reference it does not send the value it sends the variables address so it knows where to find it.

Demo 7.2: How to make parameters call by ref.

7.3 The C# out Parameters:

The *out* parameter is very similar to the *ref* keyword. You are allowed only to return one variable from a function. The *out* keyword gives the ability to return multiple values. You must use *out* in the same way as you would the *ref* keyword. Except in one case:

Demo 7.3: How to return multiple values using out parameters.

The Optional Parameters:

By default, all parameters of a method are required. But in C# 4.0, the concept of optional parameters was introduced that allows developers to declare parameters as optional.

That means, if these arguments are not passed, they will be ommitted from the execution. Optional parameters are not mandatory.

```
public int Size(int Height, int Width=1)
      {
         return Height * Width;
      }
```

```
static void Main()
      {
         MyMath m = new MyMath();
       Console.WriteLine(  m.Size(80));
         Console.ReadLine();
      }
```

Named Parameter:

Named Parameters allow developers to pass a method arguments with parameter names. Its useful while using multiple optional parameter.

```
public string Size(int Height, int Width=1,string msg="The Size is: ")
      {
         return msg + Height * Width;
      }
```

```
static void Main()
      {
         MyMath m = new MyMath();
       Console.WriteLine(  m.Size(80,msg:"The Answer is"));
         Console.ReadLine();
      }
```

How to pass array to a function as parameter.

Traditional way:

```
public void Marks(int[] marks)
    {
        foreach (int temp in marks)
        {
            Console.WriteLine(temp);
        }
    }
```

```
static void Main()
    {
        MyMath m = new MyMath();
        int[] mymarks = {50,60,70,80 };
        m.Marks(mymarks);
        Console.ReadLine();
    }
```

Using params keyword:

```
public void Marks(params int[] marks)
    {
        foreach (int temp in marks)
        {
            Console.WriteLine(temp);
        }
    }
```

```
static void Main()
    {
        MyMath m = new MyMath();
        m.Marks(50, 60, 70, 80);
        Console.ReadLine();
    }
```

7.4 C# Scope of variables:

Variable scope is an crucial subject in any programming language. One of

the most difficult errors to debug are runtime bugs. Compilation errors are easy to pin point because the compiler tells you were the error is. A runtime bug is a bug that produces incorrect results. The program runs fine but it gives unexpected results or behavior. Trying to track variable values is a daunting task if you have hundreds or thousands of variables. If you keep track of your variable scopes you will have less headaches in the future.

1.Application Level Variables.

2.Instance Level Variables (Class level/Structure Level).

3.Function Level Variables (parameters or declared inside function).

4.Block Level Variables (declared inside if or for etc.)

7.5 C# Main() Function Versions:

Functions are able to accept and return parameters. Main() is no different than any other function. The C# Main function is slightly different in that it is able to accept and return values to external applications. There are four versions of the Main() function:

static void Main() - accepts no parameters and returns no values

static void Main(string[] args) - returns no parameters and accepts a string array named args

static int Main() - returns an *int* value and accepts to parameters

static int Main(string[] args) - returns an *int* and accepts a string array named args

7.6 C# Function Overloading:

Overloading is the idea of having multiple functions that are the same name. Each function must accept different types or amount of types. This is how the compiler differentiates from overloaded functions.

Demo 7.6 : To overload function to make sum of two numbers and to concat two strings

7.7 C# Recursion

C# recursion is the idea of a function calling itself. This may seem weird but this is a good feature to produce some elegant results. Recursion is common in advanced math and science where complicated calculations require recursion. We will provide a simple and easy example of C# recursion.

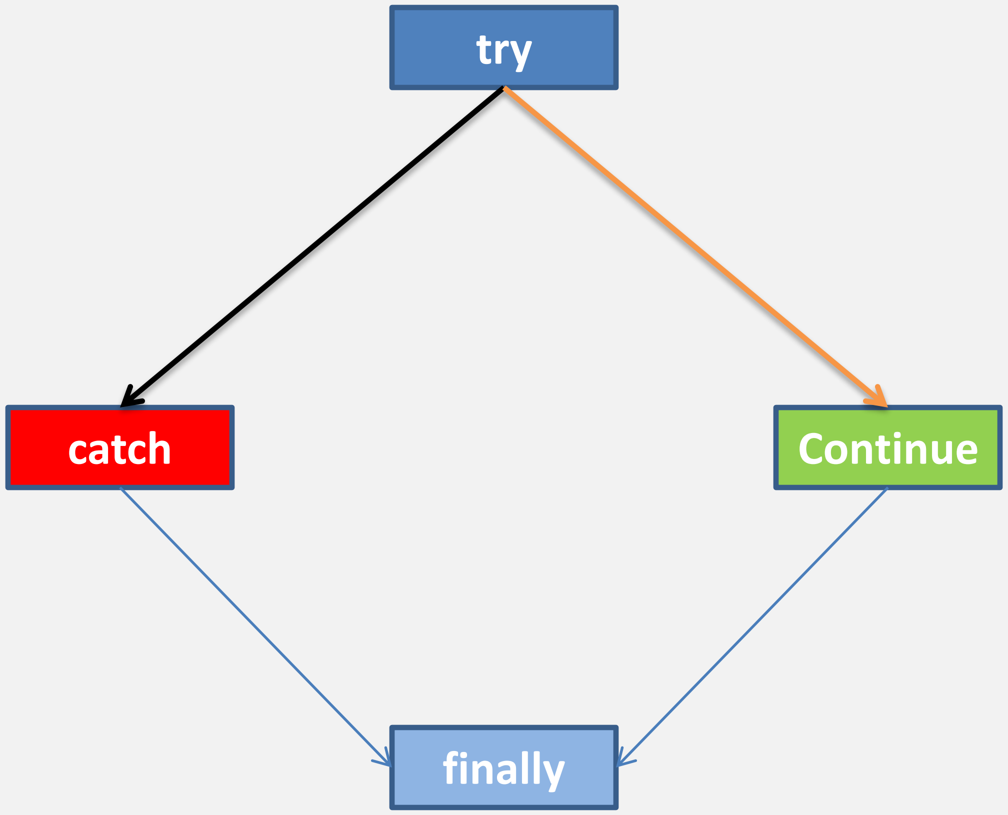
Demo 7.7 : How to increment a number by 1 using recursive calling of function as we do in Looping construct.

7.8 Exceptions and Exception Handling (System.Exception)

An exception is a response to an abnormal or unexpected condition occurring in a program, due to which the execution of a program may be aborted.

C#.NET uses a structured mechanism to handle exceptions - the *Try..Catch* block.

A Try Catch block



How to use multiple catch blocks

```
static void Main()
        {
            try
            {
                int a, b;
                a = Convert.ToInt32(Console.ReadLine());
                b = Convert.ToInt32(Console.ReadLine());
                  int c = a / b;
                Console.WriteLine("Answer is :{0}", c);
            }
            catch (DivideByZeroException ex)
            {
                Console.WriteLine(ex.Message);
            }
            catch (FormatException ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                Console.WriteLine("I am in finally");
                GC.Collect();
            }}
```

Minutes of Chapter

- ❑ Intro. To Functions in C#.
- ❑ To Return value from Function.
- ❑ What is Call by value and Call by ref ?
- ❑ What is out Parameter?
- ❑ How to define Optional Parameter?
- ❑ What is named Parameter?
- ❑ What is Function Overloading?
- ❑ What is Recursive Function?
- ❑ What are the Exceptions and How to Handling them?