C# - METHODS

A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.

To use a method, you need to –

- Define the method
- Call the method

Defining Methods in C#

When you define a method, you basically declare the elements of its structure. The syntax for defining a method in C# is as follows –

```
<Access Specifier> <Return Type> <Method Name>(Parameter List) {
   Method Body
}
```

Following are the various elements of a method –

- **Access Specifier** – This determines the visibility of a variable or a method from another class.
- **Return type** – A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is **void**.
- **Method name** – Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
- **Parameter list** – Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- **Method body** – This contains the set of instructions needed to complete the required activity.

Example

Following code snippet shows a function *FindMax* that takes two integer values and returns the larger of the two. It has public access specifier, so it can be accessed from outside the class using an instance of the class.

```
class NumberManipulator {

   public int FindMax(int num1, int num2) {
      /* local variable declaration */
      int result;

      if (num1 > num2)
         result = num1;
      else
```

```
         result = num2;

      return result;
   }
   ...
}
```

Calling Methods in C#

You can call a method using the name of the method. The following example illustrates this –

Live Demo

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public int FindMax(int num1, int num2) {
         /* local variable declaration */
         int result;

         if (num1 > num2)
            result = num1;
         else
            result = num2;
         return result;
      }

      static void Main(string[] args) {
         /* local variable definition */
         int a = 100;
         int b = 200;
         int ret;
         NumberManipulator n = new NumberManipulator();

         //calling the FindMax method
         ret = n.FindMax(a, b);
         Console.WriteLine("Max value is : {0}", ret );
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result –

```
Max value is : 200
```

You can also call public method from other classes by using the instance of the class. For example, the method *FindMax* belongs to the *NumberManipulator* class, you can call it from another class *Test*.

Live Demo

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public int FindMax(int num1, int num2) {
         /* local variable declaration */
         int result;

         if(num1 > num2)
            result = num1;
         else
            result = num2;

         return result;
      }
   }
   class Test {
      static void Main(string[] args) {
         /* local variable definition */
         int a = 100;
         int b = 200;
         int ret;
         NumberManipulator n = new NumberManipulator();

         //calling the FindMax method
         ret = n.FindMax(a, b);
         Console.WriteLine("Max value is : {0}", ret );
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result –

```
Max value is : 200
```

Recursive Method Call

A method can call itself. This is known as **recursion**. Following is an example that calculates factorial for a given number using a recursive function –

Live Demo

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public int factorial(int num) {
         /* local variable declaration */
         int result;
         if (num == 1) {
            return 1;
         } else {
            result = factorial(num - 1) * num;
            return result;
         }
      }
      static void Main(string[] args) {
         NumberManipulator n = new NumberManipulator();
         //calling the factorial method {0}", n.factorial(6));
         Console.WriteLine("Factorial of 7 is : {0}", n.factorial(7));
         Console.WriteLine("Factorial of 8 is : {0}", n.factorial(8));
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result –

```
Factorial of 6 is: 720
Factorial of 7 is: 5040
Factorial of 8 is: 40320
```

Passing Parameters to a Method

When method with parameters is called, you need to pass the parameters to the method. There are three ways that parameters can be passed to a method –

Sr.No.	Mechanism & Description
1	Value parameters This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
2	Reference parameters This method copies the reference to the memory location of an argument into the formal parameter. This means that changes made to the parameter affect the argument.
3	Output parameters This method helps in returning more than one value.