METHODS IN C#

Methods are a series of statements in a code block that perform a specific task. These statements can be executed by calling the method with the required arguments.

Method Definition in C#

Methods are defined within a class or a structure. The method definition specifies the method name and the parameters types required. When calling the method, the arguments for each parameter is specified.

The syntax for method definition is given as follows:

```
AccessSpecifier ReturnType NameOfMethod (ListOfParameters)

{

        // Method Statements

}
```

In the above syntax, the AccessSpecifier provides the visibility of the method in the class. The ReturnType is the value returned by the method. The NameOfMethod is the name provided to the method as a unique identifier. ListOfParameters is the list of the parameters in the method.

Method Calling in C#

A method can be called using its name. For the method calling, the name of the method is followed by parenthesis, even if there are no arguments.

The syntax for method calling is given as follows:

NameOfMethod (ListOfParameters);

In the above syntax, the NameOfMethod is the name provided to the method as a unique identifier. ListOfParameters is the list of the parameters in the method.

EXAMPLE:

```
// C# program to illustrate
// method calling
using System;
namespace ConsoleApplication1 {

class DemoMethods {

        // Here Sum() method asks for two
        // parameters from the user and
        // calculates the sum of these
        // and finally returns the result.
        static int Sum(int x, int y)
        {

                        // there are two local variables
                        // 'a' and 'b' where 'a' is assigned
                        // the value of parameter 'x' and
                        // 'b' is assigned the value of
                        // parameter 'y'
                        int a = x;
                        int b = y;

                        // The local variable calculates
                        // the sum of 'a' and 'b'
                        // and returns the result
                        // which is of 'int' type.
                        int result = a + b;

                        return result;
                }

        // Main Method
        static void Main(string[] args)
        {
                int a = 12;
                int b = 23;

                // Method Sum() is invoked and
                // the returned value is stored
                // in the local variable say 'c'
```

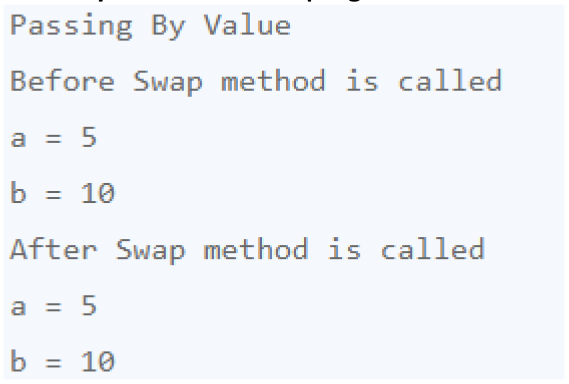
```
                        int c = Sum(a, b);

                        // Display Result
                        Console.WriteLine("The Value of the sum is " + c);
            }
}
}
```

Passing By Value

When the arguments/parameters are passed by value to a method, new memory variables are created for these arguments/parameters. The changes done in those memory variables don't reflect back to the original values.

```
using System;
namespace PassByValueDemo
{
  class Example {
    static void Swap(int m, int n)
    {
      int temp;
      temp = m;
      m = n;
      n = temp;
    }
    static void Main(string[] args)
    {
      int a = 5;
      int b = 10;
      Console.WriteLine("Passing By Value");
      Console.WriteLine("Before Swap method is called");
      Console.WriteLine("a = {0}", a);
      Console.WriteLine("b = {0}", b);
      Swap(a, b);
      Console.WriteLine("After Swap method is called");
      Console.WriteLine("a = {0}", a);
      Console.WriteLine("b = {0}", b);
    }}}
```

The output of the above program is as follows:



```
Passing By Value

Before Swap method is called

a = 5

b = 10

After Swap method is called

a = 5

b = 10
```

In the above output, you can see the values won’t get swapped.

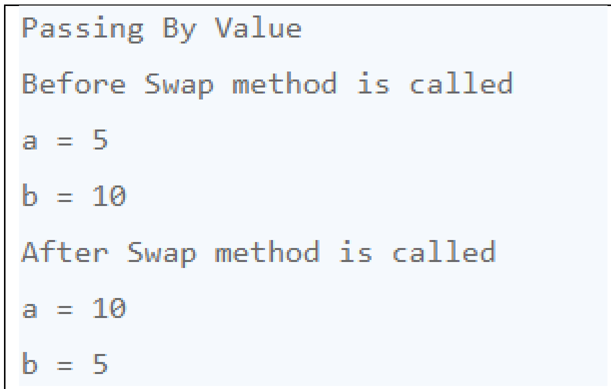
Passing By Reference

When the parameters are passed by reference to a method, new memory variables are not created for these parameters. Rather, the reference parameters point to the original parameters so that the changes done in the reference variables reflect back to the original values.

The reference parameters can be declared using the **ref** keyword in C#

```
using System;
namespace PassByRefDemo
{
  class Example {
    static void Swap(ref int m, ref int n)
    {
      int temp;
      temp = m;
      m = n;
      n = temp;
    }
    static void Main(string[] args)
    {
      int a = 5;
      int b = 10;
      Console.WriteLine("Passing By Reference");
      Console.WriteLine("Before Swap method is called");
      Console.WriteLine("a = {0}", a);
      Console.WriteLine("b = {0}", b);
      Swap(ref a, ref b);
      Console.WriteLine("After Swap method is called");
      Console.WriteLine("a = {0}", a);
      Console.WriteLine("b = {0}", b);
    }
  }
}
```

The output of the above program is as follows:



```
Passing By Value

Before Swap method is called

a = 5

b = 10

After Swap method is called

a = 10

b = 5
```

In the above output, you can see the values get swapped successfully.

Example Program Without Parameters & Without Return Type

```
// C# program to illustrate method Without
// Parameters & Without Return Type
using System;
namespace ConsoleApplication2 {
class demo1 {

        // Here the method 'PrintSentence()'
        // neither takes any parameter nor
        // returns any value. It simply performs
        // the required operations and prints
        // the result within it.
        static void PrintSentence()
        {

                Console.WriteLine("No parameters and return type void");
        }

        // Main Method
        static void Main(string[] args)
        {

                // Method Invoking or Method calling
                PrintSentence();
        }
}
}
```

Output :

```
No parameters and return type void
```

Example Program Without Parameters & With Return Value Type

```
// C# program to illustrate the method Without
// Parameters & With Return Value Type
using System;
namespace ConsoleApplication3 {

class demo2 {

        // This method takes no parameter,
        // however returns the result obtained
        static int sum()
        {
                int a = 78, b = 70, add;
                add = a + b;
                return add;
        }

        // Main Method
        static void Main(string[] args)
        {

                // Here the calling variable
                // is 'getresult'
                int getresult = sum();

                // Printing the value of
                // 'getresult' variable
                Console.WriteLine(getresult);
        }
}
}
```

Output :

```
148
```

Example Program With Parameters & Without Return Value Type

```
// C# program to illustrate Method With
// Parameters & Without Return Value Type
using System;
namespace ConsoleApplication3 {
class demo3 {

        // This method take the side of
        // the square as a parameter and
        // after obtaining the result,
        // it simply print it without
        // returning anything..
        static void perimeter(int p)
        {

                // Displaying the perimeter
                // of the square
                Console.WriteLine("Perimeter of the Square is " + 4 * p);
        }

        // Main Method
        static void Main(string[] args)
        {

                // side of square
                int p = 5;

                // Method invoking
                perimeter(p);
        }
}
}
```

Output :

```
Perimeter of the Square is 20
```

Example Program With Parameters & With Return Value Type

```
// C# program to illustrate Method With
// Parameters & With Return Value Type
using System;
namespace ConsoleApplication4 {
class demo4{

	// This method asks a number from
	// the user and using that it
	// calculates the factorial
	// of it and returns the result
	static int factorial(int n)
	{
		int f = 1;

		// Method to calculate the
		// factorial of a number
		for (int i = 1; i<= n; i++)
		{
			f = f * i;
		}

		return f;
	}

	// Main Method
	static void Main(string[] args)
	{
		int p = 4;

		// displaying result by calling the function
		Console.WriteLine("Factorial is : " + factorial(p));
	}
}
}
```

Output :

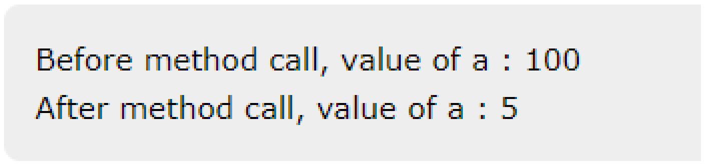
```
Factorial is : 24
```

OUTPUT PARAMETERS

Multiple return not possible in C# using return keyword. But it can done indirectly using OUTPUT parameters.

A return statement can be used for returning only one value from a function. However, using output parameters, you can return two values from a function. Output parameters are similar to reference parameters, except that they transfer data out of the method rather than into it.

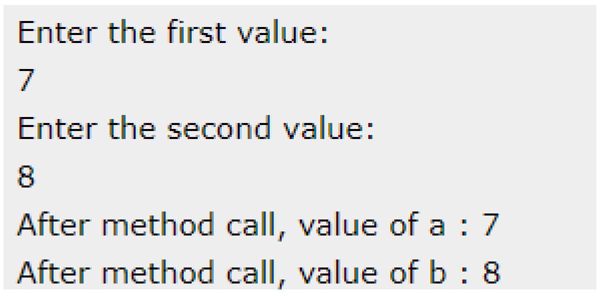
```
using System;

namespace CalculatorApplication {
  class NumberManipulator {
    public void getValue(out int x ) {
      int temp = 5;
      x = temp;
    }
    static void Main(string[] args) {
      NumberManipulator n = new NumberManipulator();

      /* local variable definition */
      int a = 100;

      Console.WriteLine("Before method call, value of a : {0}", a);

      /* calling a function to get the value */
      n.getValue(out a);

      Console.WriteLine("After method call, value of a : {0}", a);
      Console.ReadLine();
    }
  }
}
```



```
Before method call, value of a : 100
After method call, value of a : 5
```

The variable supplied for the output parameter need not be assigned a value. Output parameters are particularly useful when you need to return values from a method through the parameters without assigning an initial value to the parameter.

```
using System;

namespace CalculatorApplication {
  class NumberManipulator {
    public void getValues(out int x, out int y ) {
       Console.WriteLine("Enter the first value: ");
       x = Convert.ToInt32(Console.ReadLine());

       Console.WriteLine("Enter the second value: ");
       y = Convert.ToInt32(Console.ReadLine());
    }
    static void Main(string[] args) {
      NumberManipulator n = new NumberManipulator();

      /* local variable definition */
      int a , b;

      /* calling a function to get the values */
      n.getValues(out a, out b);

      Console.WriteLine("After method call, value of a : {0}", a);
      Console.WriteLine("After method call, value of b : {0}", b);
      Console.ReadLine();
    }
  }
}
```



```
Enter the first value:
7
Enter the second value:
8
After method call, value of a : 7
After method call, value of b : 8
```