



Zakho Technical Institute
Department of Information Technology

Object Oriented Programming

3. C# Methods (Functions)

Lecturer:

Sipan M. Hameed

www.sipan.dev

2024 - 2025

Contents

Method(function)	4
Method Declaration and Syntax:	4
Method declaration:.....	5
Calling (invoking) the method:.....	5
Methods Parameters	7
Multiple parameters.....	8
Default Parameter Value	8
Return Values	9
Named Arguments.....	11
Example: method overloading.....	12
Passing parameters to methods	13
Practical examples:.....	15
mathematical expressions	15
Geometry: shapes and solids	16
factorial	17
Exponents and powers	17
Exercise – 1.....	18
Exercise – 2 - Multiple Parameters.....	19
Exercise – 3 - Default Parameter Value	20
Exercise – 4 - Return Values	21

Exercise – 5 Method Overloading.....	22
Example – 1 - methods (functions) parameter Pass by value	23
Example – 2 - methods (functions) parameter Pass by Reference	24
Example - 3.....	25
Example - 4.....	26
Example - 5.....	27
Example - 6.....	28
Example - 7.....	29
Example - 8.....	30

Method(function)

A method is a code block that contains a series of statements to perform a specific task or operation. It is a fundamental building block of C# programs and is defined within classes. Generally, C# Methods are useful to improve code reusability by reducing code duplication. Suppose we have the same functionality to perform in multiple places, then we can create one method with the required functionality and use it wherever it is required in the application.

C# provides some pre-defined methods, that you are already familiar with, such as **Main()**, but you can also create your own methods to perform certain actions.

To use a method, you need to –

- Define the method
- Call the method

Method Declaration and Syntax:

In C#, a method (procedure/function) is declared using the following syntax:

```
<Access Modifier> <Return Type> <Method Name>(Parameter  
List)  
{  
    Method Body: Code to perform a specific task  
    return value; //(optional)  
}
```

- **Access Specifier** – Determines the visibility and accessibility of the method (public, private, protected, internal, etc.). It is useful in class inheritance.
- **Return type** – A method may return a value: in this case, the method is called “**function**”. The return type is the data type of the value the method returns (such as **int**, **string**, **char**, **double**, etc.). If the method is not returning any values, then the return type is **void** (in this case the method is called “**procedure**”).
- **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** – Parameters are variables that you pass to a method when you call it. They are enclosed within the parentheses in the method declaration. Parameters are optional (a method may contain no parameters).
- **Method body** – This contains the set of instructions needed to complete the required activity.

Method declaration:

```
static void print()
{
    Console.WriteLine("method executed");
}
```

Calling (invoking) the method:

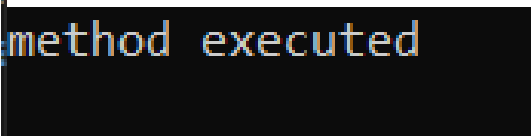
To call (execute) a method, write the method's **name** followed by two parentheses **()** and a semicolon **;**

Inside **Main()**, call the **print()** method:

```
static void Main()
{
    print();
}
```

Example:

```
class Program
{
    static void print()
    {
        Console.WriteLine("method executed");
    }
    static void Main()
    {
        print();
    }
}
```



The method can be invoked **multiple times**

```
class Program
{
    static void print()
    {
        Console.WriteLine("method excuted");
    }
    static void Main()
    {
        print();
        print();
        print();
    }
}
```

```
method executed
method executed
method executed
```

Methods Parameters

Information can be passed to methods as a parameter. Parameters act as variables inside the method.

They are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a **string** called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

```
class Program
{
    static void MyMethod(string fname)
    {
        Console.WriteLine("welcome "+ fname);
    }

    static void Main(string[] args)
    {
        MyMethod("Ahmed");
        MyMethod("Kawa");
        MyMethod("Ali");
    }
}
```

parameter

argument

```
welcome Ahmed
welcome Kawa
welcome Ali
```

When a **parameter** is passed to the method, it is called an **argument**. So, from the example above: **fname** is a **parameter**, while **Ahmed**, **Kawa** and **Ali** are **arguments**.

Multiple parameters

You can have as many parameters as you like, just separate them with commas:

```
class Program
{
    static void MyMethod(string fname, int age)
    {
        Console.WriteLine(fname+" is "+age);
    }
    static void Main(string[] args)
    {
        MyMethod("Ahmed", 25);
        MyMethod("Kawa", 22);
        MyMethod("Ali", 30);
    }
}
```

Note that when you are working with multiple parameters, the method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

Default Parameter Value

You can also use a default parameter value, by using the equals sign (=).

If we call the method without an argument, it uses the default value ("Zeravan"):

```
class Program
{
    static void MyMethod(string fname="Zeravan")
    {
        Console.WriteLine(fname);
    }
    static void Main(string[] args)
    {
        MyMethod("Ahmed");
        MyMethod();
        MyMethod("Ali");
    }
}
```

```
Ahmed
Zeravan
Ali
```


A parameter with a default value is often known as an "optional parameter". From the example above, **fname** is an optional parameter and "**Zeravan**" is the default value.

Return Values

In the previous examples, we used the **void** keyword in all examples, which indicates that the method should **not return** a value.

If you want the method to **return** a value, you can use a primitive data type (such as **int**, **double**, **string**, **char**, or **any other datatype**) instead of **void**, and use the **return** keyword inside the method:

```
class Program
```

```
{
```

```
    static int MyMethod(int x)
```

```
    {
```

```
        return 5 + x;
```

```
    }
```

```
    static void Main(string[] args)
```

```
    {
```

```
        int x=3;
```

```
        Console.WriteLine("Returned
```

```
value
```

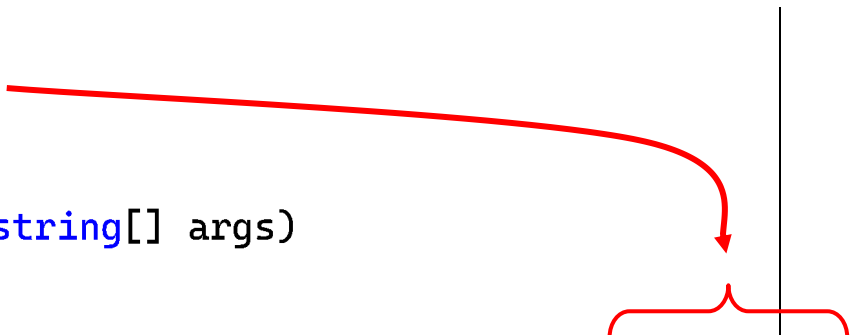
```
is:
```

```
 "+MyMethod(x));
```

```
    }
```

```
}
```

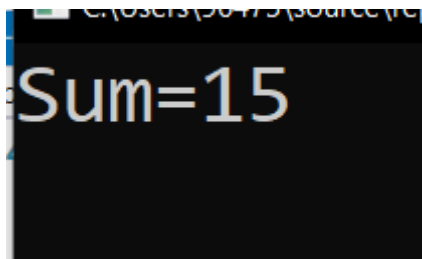
```
Returned valuse is: 8
```



Example: return the summation of the two parameter values received by the method.

```
class Program
{
    static int Sum(int x, int y)
    {
        int result = x + y;
        return result;
    }

    static void Main(string[] args)
    {
        int a = 5, b = 10;
        int res = Sum(a, b);
        Console.WriteLine("Sum="+ res);
    }
}
```



Sum=15

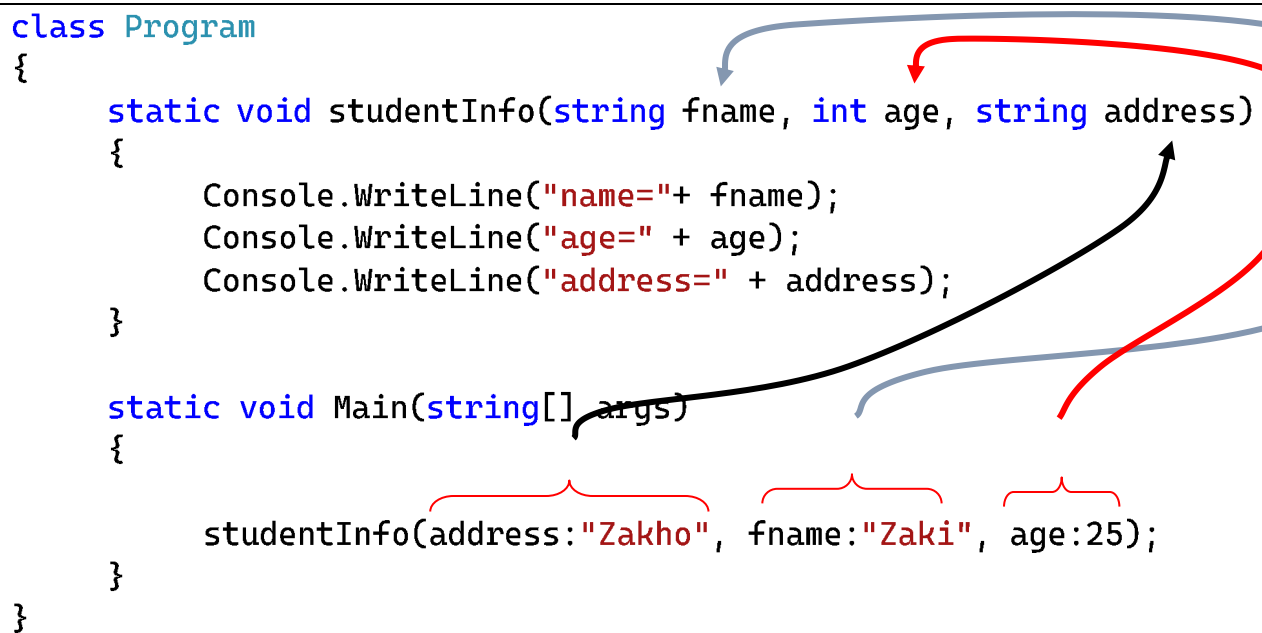
Named Arguments

It is also possible to send arguments with the **key: value** syntax.

That way, the order of the arguments does not matter:

```
class Program
{
    static void studentInfo(string fname, int age, string address)
    {
        Console.WriteLine("name="+ fname);
        Console.WriteLine("age=" + age);
        Console.WriteLine("address=" + address);
    }

    static void Main(string[] args)
    {
        studentInfo(address:"Zakho", fname:"Zaki", age:25);
    }
}
```



Method Overloading

With method overloading, **multiple methods** can have the same name with different parameters.

Function overloading allows you to define multiple methods in the same class with the same name but different parameter lists. The compiler distinguishes between these methods based on the number or types of parameters, allowing you to provide multiple implementations of a method with similar functionality.

Example: method overloading

```
class Program
{
    // Method to add two integers
    public static int Add(int a, int b)
    {
        return a + b;
    }

    // Method to add three integers
    public static int Add(int a, int b, int c)
    {
        return a + b + c;
    }

    // Method to add two doubles
    public static double Add(double a, double b)
    {
        return a + b;
    }

    static void Main()
    {
        int sum1 = Add(5, 10, 15);
        int sum2 = Add(5, 10);

        Console.WriteLine($"Sum 1: {sum1}");
        Console.WriteLine("Sum 2: "+sum2);
        Console.WriteLine("Sum 3: "+ Add(2.5, 3.7));
    }
}
```

```
Sum 1: 30
Sum 2: 15
Sum 3: 6.2
```

Passing parameters to methods

There are four different ways of passing parameters to a method in C# which are as:

1. Value
2. Ref (reference)
3. Out (reference)
4. Params (parameter arrays)

1. Passing parameter by value

By default, parameters are passed by value. In this method, a duplicate copy is made and sent to the called function. There are two copies of the variables. So if you change the value in the called method it won't be changed in the calling method.

```
Value of a is : 10  
Value of b is : 20
```

```
internal class Program  
{  
    static public void changeValues(int a, int b)  
    {  
        a = 44;  
        b = 55;  
    }  
    static void Main(string[] args)  
    {  
        int a = 10, b = 20;  
        changeValues(ref a, ref b);  
        Console.WriteLine("Value of a is : " + a);  
        Console.WriteLine("Value of b is : " + b);  
    }  
}
```

In the above code, we changed the values of data members **a** and **b** but it is not reflected back in the calling method. As the parameters are default passed by value.

2. Passing parameter by ref

Passing parameters by **reference** means passing a reference of the variable to the method. So the changes made to the parameters inside the called method will affect the original data stored in the argument variable. Using the **ref** keyword, we can pass parameters reference.

```
internal class Program
{
    static public void changeValues(ref int a, ref int b)
    {
        a = 44;
        b = 55;
    }
    static void Main(string[] args)
    {
        int a = 10, b = 20;
        changeValues(ref a, ref b);
        Console.WriteLine("Value of a is : " + a);
        Console.WriteLine("Value of b is : " + b);
        Console.ReadLine();
    }
}
```

```
Value of a is : 44
Value of b is : 55
```

Recursive Method (H.W)

- What is it?
- How does it work?
- Give an example with code.

Practical examples:

mathematical expressions

$$A = x^1 + x^2 + x^3 + \dots + x^n$$

$$B = 1! + 2! + x! + \dots + x!$$

$$C = \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \dots + \frac{x^n}{n!}$$

$$D = \frac{x^1}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} \dots - \frac{x^n}{n!}$$

$$E = \frac{n!}{r!(n-r)!}$$

$$F = \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots - \frac{x^n}{n!}$$

$$G = \frac{0!}{x^0} - \frac{2!}{x^2} + \frac{4!}{x^4} - \frac{6!}{x^6} \dots - \frac{n!}{x^n}$$

$$H = \frac{1!}{x^1} + \frac{2!}{x^2} + \frac{3!}{x^3} + \frac{4!}{x^4} \dots + \frac{n!}{x^n}$$

$$I = \frac{1!}{x^3} + \frac{2!}{x^6} + \frac{3!}{x^9} + \frac{4!}{x^{12}} \dots + \frac{n!}{x^{n*3}}$$

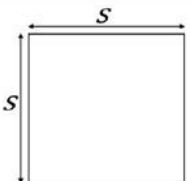
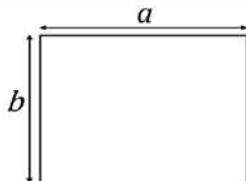
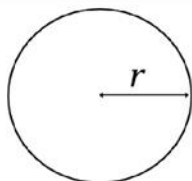
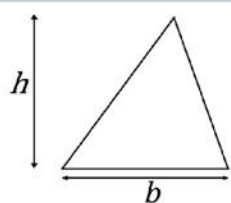
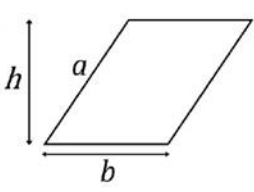
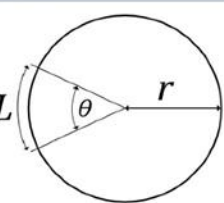
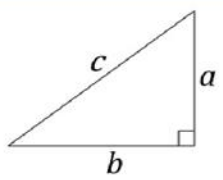
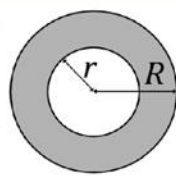
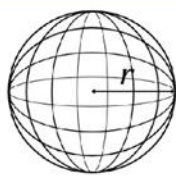
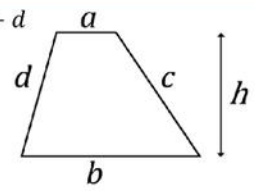
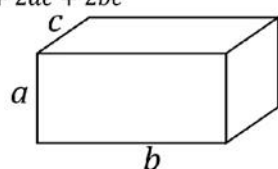
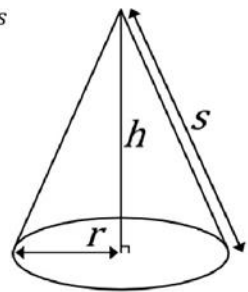
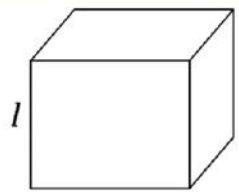
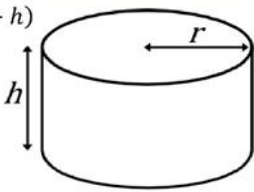
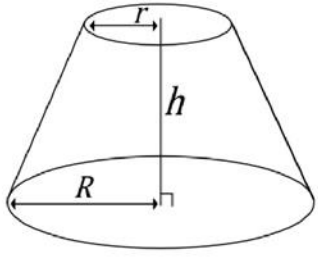
Fibonacci sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233

Geometry: shapes and solids

GEOMETRY

SHAPES AND SOLIDS

<p>SQUARE</p> $P = 4s$ $A = s^2$ 	<p>RECTANGLE</p> $P = 2a + 2b$ $A = ab$ 	<p>CIRCLE</p> $P = 2\pi r$ $A = \pi r^2$ 
<p>TRIANGLE</p> $P = a + b + c$ $A = \frac{1}{2}bh$ 	<p>PARALLELOGRAM</p> $P = 2a + 2b$ $A = bh$ 	<p>CIRCULAR SECTOR</p> $L = \pi r \frac{\theta}{180^\circ}$ $A = \pi r^2 \frac{\theta}{360^\circ}$ 
<p>PYTHAGOREAN THEOREM</p> $a^2 + b^2 = c^2$ $c = \sqrt{a^2 + b^2}$ 	<p>CIRCULAR RING</p> $A = \pi(R^2 - r^2)$ 	<p>SPHERE</p> $S = 4\pi r^2$ $V = \frac{4\pi r^3}{3}$ 
<p>TRAPEZOID</p> $P = a + b + c + d$ $A = h \frac{a+b}{2}$ 	<p>RECTANGULAR BOX</p> $A = 2ab + 2ac + 2bc$ $V = abc$ 	<p>RIGHT CIRCULAR CONE</p> $A = \pi r^2 + \pi rs$ $s = \sqrt{r^2 + h^2}$ $V = \frac{1}{3}\pi r^2 h$ 
<p>CUBE</p> $A = 6l^2$ $V = l^3$ 	<p>CYLINDER</p> $A = 2\pi r(r + h)$ $V = \pi r^2 h$ 	<p>FRUSTUM OF A CONE</p> $V = \frac{1}{3}\pi h(r^2 + rR + R^2)$ 

factorial

The list of factorial values from 1 to 10 are:

n	Factorial of a Number n!	Expansion	Value
1	1!	1	1
2	2!	2×1	2
3	3!	$3 \times 2 \times 1$	6
4	4!	$4 \times 3 \times 2 \times 1$	24
5	5!	$5 \times 4 \times 3 \times 2 \times 1$	120
6	6!	$6 \times 5 \times 4 \times 3 \times 2 \times 1$	720
7	7!	$7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$	5,040
8	8!	$8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$	40,320
9	9!	$9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$	362,880
10	10!	$10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$	3,628,800

Factorial of 5 can be calculated as:

$$5! = 1 \times 2 \times 3 \times 4 \times 5$$

$$5! = 120$$

Therefore, the value of factorial of 5 is 120.

Exponents and powers

$$2^2 = 2 \text{ raised to power } 2 = 2 \times 2 = 4$$

$$5^3 = 5 \text{ raised to power } 3 = 5 \times 5 \times 5 = 125$$

Exercise – 1

The following example has a method that takes a `string` called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

```
using System;

namespace MyApplication
{
    class Program
    {
        static void MyMethod(string fname)
        {
            Console.WriteLine(fname + " Refsnes");
        }

        static void Main(string[] args)
        {
            MyMethod("Liam");
            MyMethod("Jenny");
            MyMethod("Anja");
        }
    }
}
```

Output:

```
Liam Refsnes
Jenny Refsnes
Anja Refsnes
```

Note:

When a parameter is passed to the method, it is called an argument. So, from the example above: `fname` is a **parameter**, while Liam, Jenny and Anja are **arguments**.

Exercise – 2 - Multiple Parameters

You can have as many parameters as you like, just separate them with commas:

using System;

```
namespace MyApplication
{
    class Program
    {
        static void MyMethod(string fname, int age)
        {
            Console.WriteLine(fname + " is " + age);
        }

        static void Main(string[] args)
        {
            MyMethod("Liam", 5);
            MyMethod("Jenny", 8);
            MyMethod("Anja", 31);
        }
    }
}
```

Output:

```
Liam is 5
Jenny is 8
Anja is 31
```

Exercise – 3 - Default Parameter Value

You can also use a default parameter value, by using the equals sign (=).

```
using System;

namespace MyApplication
{
    class Program
    {
        static void MyMethod(string country = "Norway")
        {
            Console.WriteLine(country);
        }

        static void Main(string[] args)
        {
            MyMethod("Sweden");
            MyMethod("India");
            MyMethod();
            MyMethod("USA");
        }
    }
}
```

Output:

```
Sweden
India
Norway
USA
```

Exercise – 4 - Return Values

If you want the method to return a value, you can use a primitive data type (such as int or double) instead of void, and use the return keyword inside the method:

```
using System;

namespace MyApplication
{
    class Program
    {
        static int MyMethod(int x)
        {
            return 5 + x;
        }

        static void Main(string[] args)
        {
            Console.WriteLine(MyMethod(3));
        }
    }
}
```

Output:

```
8
```

Exercise – 5 Method Overloading

Consider the following example, which have two methods that add numbers of different type:

```
using System;

namespace MyApplication
{
    class Program
    {
        static int PlusMethod(int x, int y)
        {
            return x + y;
        }

        static double PlusMethod(double x, double y)
        {
            return x + y;
        }

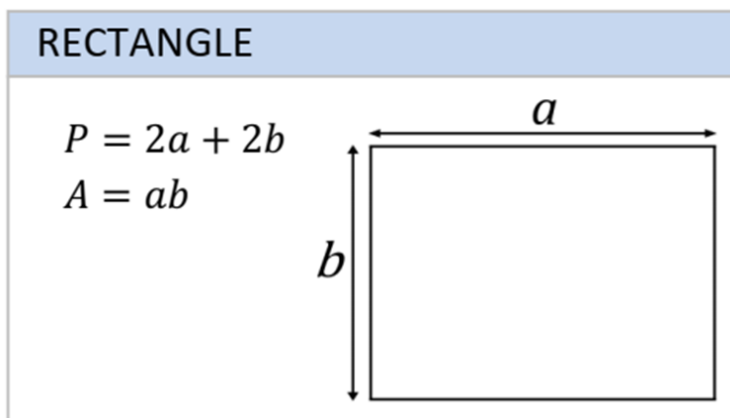
        static void Main(string[] args)
        {
            int myNum1 = PlusMethod(8, 5);
            double myNum2 = PlusMethod(4.3, 6.26);
            Console.WriteLine("Int: " + myNum1);
            Console.WriteLine("Double: " + myNum2);
        }
    }
}
```

Output :

```
Int: 13
Double: 10.559999999999999
```

Example – 1 - methods (functions) parameter Pass by value

Write a C# Sharp program using methods (functions)_to solve the following geometry equations:



Solution:

```
using System;
namespace ConsoleApp8
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int a = 4, b = 8;
            Console.WriteLine("P={0} , A = {1}",P(a,b),A(a,b));
        }

        static double P(int a ,int b)
        {
            return 2 * a + 2 * b;
        }

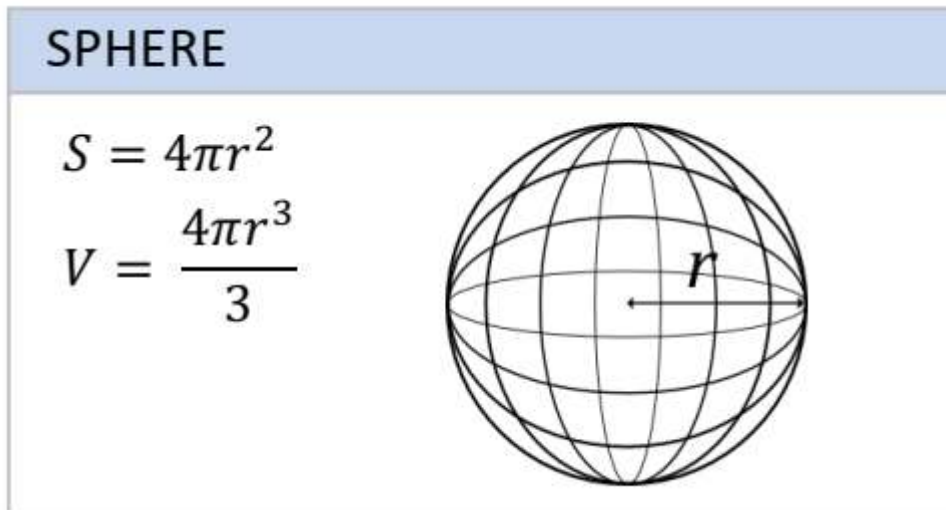
        static double A(int a ,int b)
        {
            return a * b;
        }
    }
}
```

Output:

```
P=24 , A = 32
Press any key to continue . . .
```

Example – 2 - methods (functions) parameter Pass by Reference

Write a C# Sharp program using methods (functions) **parameter Pass by Reference** to solve following geometry equations:



Solution:

```
using System;
namespace ConsoleApp8
{
    internal class Program
    {
        public const double pi = 3.14;
        static void Main(string[] args)
        {
            int r = 8;
            Console.WriteLine(" S(r) = {0}", S(ref r));
            Console.WriteLine("V(r) = {0}", V(ref r));
        }

        static double S(ref int r)
        {
            return 4 * pi * r * r;
        }
        static double V(ref int r)
        {
            return (4 * pi * r * r * r)/3;
        }
    }
}
```

Output:

```
S(r) = 803.84
V(r) = 2143.573333333333
Press any key to continue . . .
```


Example - 3

Write a C# Sharp program to find factorial of any given number:

$$\text{factorial} = n! = 1 \times 2 \times 3 \times 4 \times 5 \times \dots \times n$$

solution:

```
using System;
namespace ConsoleApp8
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" factorial of 5 = {0}", factorial(5));
            Console.WriteLine(" factorial of 8 = {0}", factorial(8));
            Console.WriteLine(" factorial of 69 = {0}", factorial(69));
        }

        static double factorial(int n)
        {
            double s = 1;
            for (int i = 1; i <= n; i++)
            {
                s = s * i;
            }
            return s;
        }
    }
}
```

Output:

```
factorial of 5 = 120
factorial of 8 = 40320
factorial of 69 = 1.71122452428141E+98
Press any key to continue . . .
```

Example - 4

Write a C# Sharp program to find power of any given number using methods (functions):

$$5^3 = 5 \text{ raised to power } 3 = 5 \times 5 \times 5 = 125$$

$$\text{Power} = x^n = x * x * x * x \dots * x$$

Solution:

```
using System;
namespace ConsoleApp8
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" 5 power 3 == {0}", power(5,3));
            Console.WriteLine(" 2 power 4 == {0}", power(2,4));
            Console.WriteLine(" 6 power 2 == {0}", power(6,2));
        }

        static double power(int x,int n)
        {
            double s = 1;
            for (int i = 1; i <= n; i++)
            {
                s = s * x;
            }
            return s;
        }
    }
}
```

Output:

```
5 power 3 == 125
2 power 4 == 16
6 power 2 == 36
Press any key to continue . . .
```

Example - 5

Write a C# Sharp program using methods (functions) to solve following equation:

$$A(x, n) = x^1 + x^2 + x^3 + \dots \dots \dots x^n$$

Solution:

```
using System;
namespace ConsoleApp8
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" A(3,4) = {0}", A(3,4));
        }

        static double A(int x,int n)
        {
            double acc = 0;
            for (int i = 1; i <= n; i++)
            {
                acc = acc + power(x, i);
            }
            return acc;
        }

        static double power(int x, int n)
        {
            double s = 1;
            for (int i = 1; i <= n; i++)
            {
                s = s * x;
            }
            return s;
        }
    }
}
```

Output:

```
A(n A(3,4) = 120
Press any key to continue . .
```

Example - 6

Write a C# Sharp program using methods (functions) to solve following equation:

$$C(x, n) = \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

Solution:

```
using System;
namespace ConsoleApp8
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" C(3,4) = {0}", C(3,4));
        }

        static double C(int x, int n)
        {
            double acc = 0;
            for (int i = 1; i <= n; i++)
            {
                acc = acc + power(x, i) / factorial(i);
            }
            return acc;
        }

        static double power(int x, int n)
        {
            double s = 1;
            for (int i = 1; i <= n; i++)
            {
                s = s * x;
            }
            return s;
        }

        static double factorial(int n)
        {
            double s = 1;
            for (int i = 1; i <= n; i++)
            {
                s = s * i;
            }
            return s;
        }
    }
}
```

Output:

```
C(3,4) = 15.375
Press any key to continue . . .
```

Example - 7

Write a C# Sharp program using methods (functions) to solve following equation:

$$F(x, n) = \frac{x^1}{1!} + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} \dots \dots \dots + \frac{x^n}{n!}$$

Solution:

```
using System;
namespace ConsoleApp8
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" F(3,4) = {0}", F(3,4));
        }

        static double F(int x, int n)
        {
            double acc = 0;
            for (int i = 1; i <= n; i = i+2)
            {
                acc = acc + power(x, i) / factorial(i);
            }
            return acc;
        }
        static double power(int x, int n)
        {
            double s = 1;
            for (int i = 1; i <= n; i++)
            {
                s = s * x;
            }
            return s;
        }
        static double factorial(int n)
        {
            double s = 1;
            for (int i = 1; i <= n; i++)
            {
                s = s * i;
            }
            return s;
        }
    }
}
```

Output:

```
F(3,4) = 7.5
Press any key to continue . . .
```

Example - 4

Example - 8

Write a C# Sharp program using methods (functions) to solve following equation:

$$K(x, n) = \frac{1!}{x^3} + \frac{2!}{x^6} + \frac{3!}{x^9} + \frac{4!}{x^{12}} + \dots + \frac{n!}{x^{n*3}}$$

Solution;

```
using System;
namespace ConsoleApp8
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" K(3,4) = {0}", K(3,4));
        }

        static double K(int x, int n)
        {
            double acc = 0;
            for (int i = 1; i <= n; i = i+2)
            {
                acc = acc + factorial(i)/power(x,i*3);
            }
            return acc;
        }

        static double power(int x, int n)
        {
            double s = 1;
            for (int i = 1; i <= n; i++)
            {
                s = s * x;
            }
            return s;
        }

        static double factorial(int n)
        {
            double s = 1;
            for (int i = 1; i <= n; i++)
            {
                s = s * i;
            }
            return s;
        }
    }
}
```

Output:

```
K(3,4) = 0.0373418686175888
Press any key to continue . . .
```