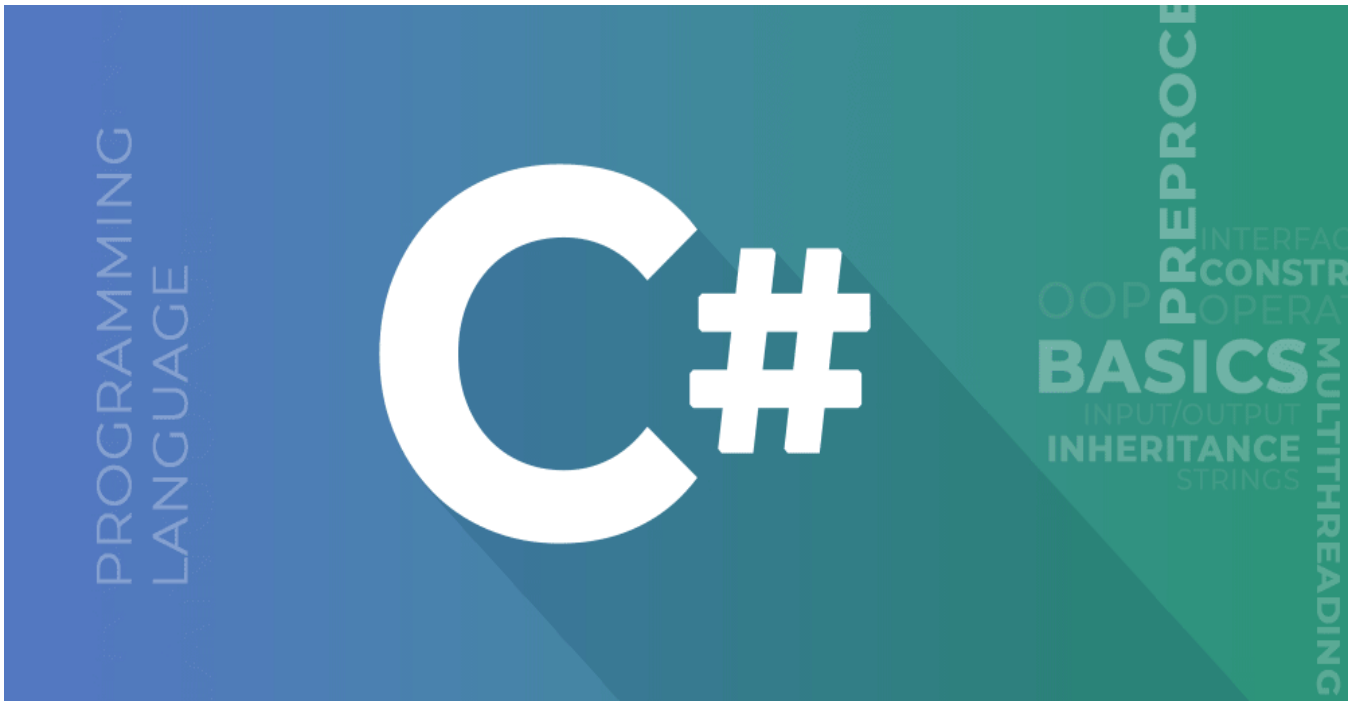


C# (C Sharp) Tutorial

Provided by

Muhammad Farhan

Contact No: +92316-2859445

Email: mohammadfarhan44500@gmail.com

GitHub: https://github.com/muhammadfarhandevelper

INTRODUCTION TO C# PROGRAMMING LANGUAGE

Microsoft introduced a new language called C# (C Sharp).

C# is designed to be a simple, modern, general-purpose, object-oriented programming language, borrowing key concepts from several other languages, like Java.

C# was developed by **Anders Hejlsberg** and his team during the development of **.Net Framework.**

C# is a part of .NET Framework and currently in version 4.8.1

WE CAN USE C# FOR BUILDING VARIETY OF APPLICATIONS.

- **WINDOWS APPLICATION:** using **console** application or **winform** application.
- **MOBILE APPLICATIONS:** for phones such as Nokia Lumia (built-in support) but we can use a third party tool or library called **"XAMARIN"** to create mobile applications for ANDROID and IOS as well.
- **WEB APPLICATON:** using ASP.NET web forms or ASP.NET MVC.
- **GAMING APPLICATION:** Unity.

C# could theoretically be compiled to machine code, but in real life, it's always used in combination with the .NET framework. Therefore, applications written in C#, requires the .NET framework to be installed on the computer running the application. While the .NET framework makes it possible to use a wide range of languages, C# is sometimes referred to as THE .NET language, perhaps because it was designed together with the framework.

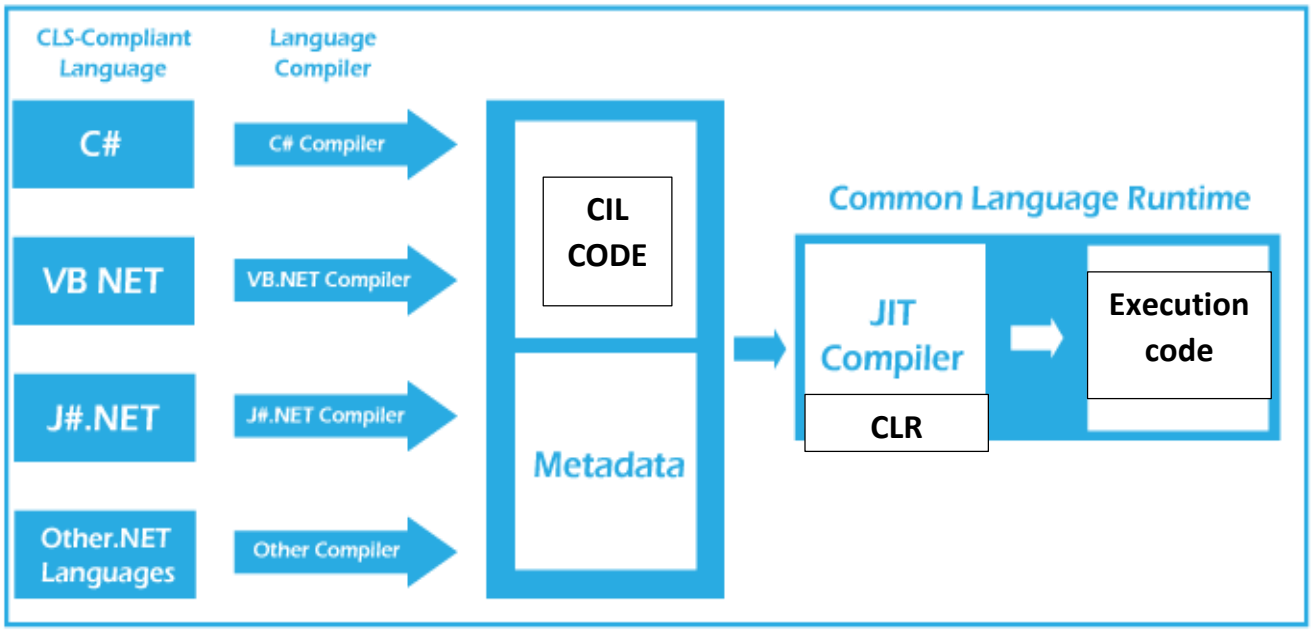
WHAT IS MAIN METHOD:

Main method is the entry point of your application.

What is CLR:

- Is the foundation of the .NET Framework.
- Acts as an execution engine for the .NET Framework.
- Manages the execution of programs and provides a suitable environment for programs to run.

- Provides a multi-language execution environment.
- Is a backbone of .NET Framework



Exceution of a .NET Application

Console Write method:

WriteLine()

Writes the text representation of the specified objects, followed by the current line terminator, to the standard output stream using the specified format information.

Example:

```
Console.WriteLine("This is Message ");
```

Write()

This method uses the composite formatting feature of .NET to convert the value of an object to its text representation and embed that representation in a string.

Example:

```
Console.Write("This is Message ");
```

ReadLine()

This method is used to read the next line of characters from the standard input stream.

Example:

```
string name = Console.ReadLine();
```

Another Example:

```
Console.WriteLine("Enter Marks : ");
int marks = int.Parse(Console.ReadLine());
```

ReadKey()

ReadKey() Obtains the next character or function key pressed by the user.

Example:

```
Console.ReadKey();
```

Concatenation:

```
string name1 = "Muhammad";
string name2 = "Farhan";
Console.WriteLine("Your Name is " + name1 + name2);
```

Placeholder:

```
string name1 = "Muhammad";
string name2 = "Farhan";

Console.WriteLine("Your First Name {0} last name {1}",name1,name2);
```

Assignment: Take Marks of student and sum it.

Datatypes of C#:

Integral Data Types:

Signed Integers: which takes negative and positive values.

Unsigned Integers: which only takes positive values.

Datatypes	Memory Size	Range
short	2 byte	-32,768 to 32,767
signed Short	2 byte	-32,768 to 32,767
unsigned Short	2 byte	0 to 65,535
int	4 byte	-2,147,483,648 to 2,147,483,647
signed int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned int	4 byte	0 to 4,294,967,295
long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
signed long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long	8 byte	0 to 18,446,744,073709,551,615
float	4 byte	$1.5 * 10^{-45}$ to $3.4 * 10^{38}$ (7 Digit)
double	8 byte	$5 * 10^{-324}$ to $1.7 * 10^{308}$
decimal	16 byte	$-7.9 * 10^{28}$ to $7.9 * 10^{28}$

Example:

```
sbyte a = 127;
byte b = 255;

short c = 32767;
ushort d = 65535;

int e = 2147483647;
uint f = 4294967295;

Console.WriteLine(a);
```

```
Console.WriteLine(b);
Console.WriteLine(c);
Console.WriteLine(d);
Console.WriteLine(e);
Console.WriteLine(f);
```

MaxValue and MinValue:

Take Length of any data types.

```
Console.WriteLine(int.MaxValue);
Console.WriteLine(int.MinValue);
```

Bool Data Types:

Bool data type take only true and false.

Example:

```
  int a = 20;
int b = 23;

bool c = a > b;
```

Float Double and Decimal Data Type:

C#	.Net Type	Size	Precision
Float	System.Single	4 bytes	7 digits
Double	System.Double	8 bytes	15 – 16 digits
decimal	System.Decimal	16 bytes	28-29 decimal places

Example:

```
float a = 235.6767f;
double b = 46767678.4554d;
decimal c = 45546456.45645664565m;

Console.WriteLine(a);
Console.WriteLine(b);
Console.WriteLine(c);
```

String and Character Datatypes:

- String stores multiple characters in a single variable.
- Double quotes will be used with string data types.
- Char stores single character at a time in a variable.
- Single quotes will be used for char data type.

Example:

```
char a = 'A';
string b = "Farhan";

Console.WriteLine(a);

Console.WriteLine(b);
```

Escape Sequences:

Character	Escape Sequence name
\'	Single quote
\"	Double quote
\\	Backslash
\0	Null
\a	Alert
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical quote

Example:

```
Console.WriteLine(" \" welcome   \" ");
Console.WriteLine(" \' welcome   \' ");
Console.WriteLine(" \\ welcome   \\ ");
Console.WriteLine(" welcome   \n ");
Console.WriteLine(" \t\t\t welcome   \n ");
```

For Sound of Alert:

```
Console.WriteLine(" \a ");
```

Verbatim Literal:

- Verbatim literal is a string with an @symbol.
- Verbatim literal make escape sequence translate as normal printable character to enhance readability.

Example:

```
string path = @"D:\\Farhan\\newfolder\filepath";

Console.WriteLine(path);
```

Data Types Conversion in C#:

There are two types of conversion implicit and explicit Conversion.

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 exception during the conversion.

Example:

```
int a = 4666;
float b = a;
Console.WriteLine(b);
```

Explicit Conversion:

In this data will be loss.

Example:

```
float a = 446.45f;

int b = (int)a;

Console.WriteLine(b);
```

Example:

```
float a = 446.45f;

int b = Convert.ToInt32(a);

Console.WriteLine(b);
```

Another Example:

```
    string a = "34";
string b = "40";

int c = Convert.ToInt32(a) + Convert.ToInt32(b);

Console.WriteLine(c);
```

Constant:

- A constant has a fixed value that remains unchanged throughout the program.
- In C#, you can declare constants for all data types.
- You have to initialize a constant at the time of its declaration.
- Constant are used for value types rather than for reference types.
- To declare an identifier as a constant the “const” keyword.

Example:

```
const int a = 34;

Console.WriteLine(a);
```

C# Operator:

Operators are used to perform operations on variables and values.

Example:

a + b

a, b = operand

+ = operator

Arithmetic Operator:

- Arithmetic Operator are **binary operators** because the work with **two operands**, with the operator being placed in between the operands.
- These operators allow you to perform computations on numeric or string data.

Example:

- **+**
- **-**
- *****
- **/**
- **%**

```
int a = 5;
int b = 3;

int c = a + b;
int d = a - b;
int e = a * b;
int f = a / b;
int g = a % b;


Console.WriteLine("Your Sum : " + c);
Console.WriteLine("Your Sub : " + d);
Console.WriteLine("Your Multiplication : " + e);
```

```
Console.WriteLine("Your Division : " + f);
Console.WriteLine("Your Remainder: " + g);
```

Assignment Operators:

- Assignment operators are used to assign the value of the right side operand to the operand on the left side using the equal to operator (=).
- The assignment operators are as follows:

= , += , -= , *= , /= , %=

Example:

```
int a = 5;

a += 3;

Console.WriteLine(a);
```

Another Example:

```
int a = 5;

a *= 3;

Console.WriteLine(a);
```

Relational or Comparison Operator:

- Relational operators make a comparison between two operands and return a Boolean value, true or false.
- ==
- !=
- >
- <
- >=
- <=

```
int a = 5;
int b = 3;
```

```
bool c =    a > b;
bool d = a < b;
bool e = a >= b;
bool f = a <= b;
bool g = a == b;
bool h = a != b;

Console.WriteLine(c);
Console.WriteLine(d);
Console.WriteLine(e);
Console.WriteLine(f);
Console.WriteLine(g);
Console.WriteLine(h);
```

Logical Operator:

- Logical operators perform Boolean logical operations on both the operands. They return a Boolean values based on the logical operator used.
- There are three types of Logical Operator:
- &&
- ||
- !

Example:

```
int a = 5;
int b = 3;
int c = 6;

bool d = !(a > b && a > c);

Console.WriteLine(d);
```

Increment and Decrement Operators:

- Two of the most common calculations performed in programming are increasing and decreasing the value of the variable by 1.
- In C#, the increment **(++)** is used to increase the value by 1 while the decrement operator **(--)** is used to decrease the value by 1.

- If the operators placed before the operand, the expression is called **pre-increment** or **pre-decrement**.
- If the operator is placed after the operand, the expression is called **post-increment** or **post-increment.**

Example:

```
int a = 5;

Console.WriteLine(a++);

Console.WriteLine(a--);
```

Another Example:

```
int a = 5;

Console.WriteLine(++a);
Console.WriteLine(--a);
```

Ternary Operator:

- C# includes a special type of decision making operator ?: called the ternary operator.

Syntax:

Boolean expression ? true statement : false statement

Example:

```
int a = 5;

string c = a > 5 ? "true condition" : "false coniditon";

Console.WriteLine(c);
```

Selection Statement:

- Is a programming construct supported by C# that controls the flow of a program?

- Executes a particular block of statements based on a Boolean condition, which is an expression returning true or false.
- Allow you to take logical decisions about executing different block of a program to achieve the required logical output.

C# support the following decision making constructs.

- If, else
- If, else, if
- Nested if
- Switch case

Example:

```
int a = 5;
int b = 3;

if(a > b)
{
   Console.WriteLine("A is greater than B");
}
else
{
   Console.WriteLine("B is greater than A");
}
```

Another Example:

```
Console.Write("Enter Username : ");
string username = Console.ReadLine();

Console.Write("Enter Password : ");
string password = Console.ReadLine();


if (username == "farhan" && password == "123")
{
   Console.WriteLine("Login Successfully ");
}
else
```

```
{
   Console.WriteLine("Incorrect Username and Password!");

}
```

The IF Else IF Statement:

- The if else if construct allows you to check multiple conditions to execute a different block of code for each condition.
- It is also referred to as if else if ladder.
- The construct start with the if statement followed by multiple else if statements followed by an optional else block.

Example:

```
int per = 90;

if (per >= 90)
{
   Console.WriteLine("Grade is A+1");
}
else if (per >= 80)
{
   Console.WriteLine("Grade is A+");
}
else if (per >= 70)
{
   Console.WriteLine("Grade is A");
}
else if (per >= 60)
{
   Console.WriteLine("Grade is B");
}
else if (per >= 50)
{
   Console.WriteLine("Grade is C");
}
else if (per >= 40)
{
   Console.WriteLine("Grade is D");
```

```
}
else
{
   Console.WriteLine("Fail !");

}
```

Nested if:

- The nested if construct consists of multiple if statements.
- The nested if construct starts with the if statements, which is called the **outer if** statements, and contains multiple if statements, which are called **inner if** statements.

Example:

```
int biryanirate = 300;
int zardarate = 100;
int kormarate = 400;
int niharirate = 500;
int rotirate = 20;


Console.WriteLine("\n======== Welcome On Our Restaruant \n\n");

Console.Write("Enter Paisy ? : ");
int paisy = int.Parse(Console.ReadLine());


if (paisy >= 1000)
{

   Console.WriteLine("\n1) Pakistani Dishes \t\t\t 2) Cheese Dished \n 3)Turkish Dishes \t\t\t  4) Italian Dished ");
   Console.Write("Chooose Option : ");
   int option = int.Parse(Console.ReadLine());

   if (option == 1)
   {
      Console.WriteLine("1) Biryani \n 2) Zarda \n 3) Korma \n 4) Nihari \n 5) Roti");

      Console.Write("Chose Option : ");
```

```
            int op = int.Parse(Console.ReadLine());

            if (op == 1)
            {
                paisy = paisy - biryanirate;

                Console.WriteLine("\n\n Billing \n Biryani : {0} \n Balance Amount : {1}",
biryanirate, paisy);
            }

        }
    }

    else
    {
        Console.WriteLine("\n\n \a Beta Ghar Jao ");
    }
```

Switch Case:

- A program is difficult to comprehend when there are too many if statements representing multiple selection constructs.

Example:

```
    int week_number = 3;


    switch (week_number)
    {

        case 1:
            Console.WriteLine("Monday");
            break;
        case 2:
            Console.WriteLine("Tuesday");
            break;
        case 3:
            Console.WriteLine("wednesday");
            break;
        case 4:
```

```
            Console.WriteLine("Thursday");
            break;
        default:
            Console.WriteLine("No Day ");
            break;

    }
```

Assignment: Mark sheet

Loop Constructs:

- Loop allow you to execute a single statement or a block of statements repetitively.
- The most common uses of loops include displaying a series of numbers and taking repetitive input.
- In **software programming,** a loop construct contains a condition that helps the compiler identify the number of times a specific block will be executed.

The loop constructs are also referred.

C# support four types of loop constructs such as:

1. For loop
2. While loop
3. Do..while loop
4. Foreach loop

For Loop:

- The for statement is similar to the while statement is its function.
- The statement within the body of the loop are executed as long as the condition is true.
- Here too, the condition is checked before the statements are executed.

There are 3 things in for loop:

- Initialization
- Condition
- Increment / Decrement

Example:

```
for (int i = 1; i <= 5; i++)
{
   Console.WriteLine(i);
}
```

Another Example (Create table using loop):

```
Console.Write("Enter Table No : ");
int tbl = int.Parse(Console.ReadLine());


for (int i = 1; i <= 10; i++)
{
   Console.WriteLine(tbl + " X " + i + " = " + i * tbl);
}
```

Another Example (Decrement Example)

```
for (int i = 10; i >= 1; i--)
{
   Console.WriteLine(i);
}
```

The While Loop:

- The while loop is used to execute a block of code repetitively as long as the condition of the loop remains true.

Example:

```
int i = 0;

while ( i <= 10)
{
   Console.WriteLine(i);
   i++;
}
```

Do While Loop:

- The do while loop is similar to the while loop; however, it is always executed at least once without the condition being checked.
- The loop starts with the do keyword and is followed by a block of executable statements.
- The while statements along with the condition appears at the end of this block.

Example:

```
int i = 0;

do
{

   Console.WriteLine("this is number : " + i);

} while (i <= 10);
```

Difference between while loop and do while loop:

While Loop	Do-while loop
This is entry controlled loop. It checks condition before entering into loop	This is exit control loop. Checks condition when coming out from loop
The while loop may run zero or more times	Do-While may run more than one times but at least once.
The variable of test condition must be initialized prior to entering into the loop	The variable for loop condition may also be initialized in the loop also.

while(condition){
//statement
}
	do{
	//statement
	}while(condition);

Nested For Loop:

- The nested for loop consists of multiple **for statements.**
- when one for loop is enclosed inside another for loop, the loops are said to be nested.

- The for loop the encloses the other for loop is referred to as the **outer for loop** whereas the enclosed for loop is referred to as the **inner for loop.**
- The outer for loop determines the number of times the inner for loop will be invoked.

Example:

```
for(int i =0; i <= 5; i++)
{

   for(int j = 0; j <=3; j++)
   {

      Console.WriteLine(i + "  " + j);

   }

}
```

Jump Statements in C#:

- Jump statements are used to transfer control from one point in a program to another.
- There will be situations where you need to exit out of a loop prematurely and continue with the program.
- In such cases, jump statements are used. Jump statements unconditionally transfer control of a program to a different location.
- The location to which a jump statements transfers control is called the target of the jump statements.

C# support four types of jump statements. These are as follows.

- Break
- Continue
- Goto
- Return

Break:

The break statement is used to terminate the loop or statement in which it present. After that, the control will pass to the statements that present after the break statement.

Example:

```
for(int i =0; i <= 10; i++)
{

   if(i == 5)
   {
      break;
   }

}


Console.WriteLine("Loop is terminated .... ");
```

Continue:

This statement is used to skip over the execution part of the loop on a certain condition. After that, it transfers the control to the beginning of the loop. Basically, it skips its following statements and continues with the next iteration of the loop.

Example:

```
for(int i =0; i <= 10; i++)
{

   if(i == 5)
   {
      continue;
   }

}


Console.WriteLine("Loop is terminated .... ");
```

goto statement:

This statement is used to transfer control to the labeled statement in the program. The label is the valid identifier and placed just before the statement from where the control is transferred.

Example:

```
for(int i =0; i <= 10; i++)
{

   if(i == 5)
   {
      goto end;
   }

}


Console.WriteLine("Loop is terminated .... ");
Console.WriteLine("Loop is terminated .... ");
Console.WriteLine("Loop is terminated .... ");
Console.WriteLine("Loop is terminated .... ");
Console.WriteLine("Loop is terminated .... ");
Console.WriteLine("Loop is terminated .... ");


end:

Console.WriteLine("program exit..");
```

Restarting Program:

```
string confirm;

do
{
   Console.WriteLine("Enter First Number ");
   int num1 = int.Parse(Console.ReadLine());

   Console.WriteLine("Enter Second Number ");
   int num2 = int.Parse(Console.ReadLine());

   int sum = num1 + num2;

   Console.WriteLine("Your addition is  : " + sum);

   Console.Write("Do you want to repeat your program ? ");
```

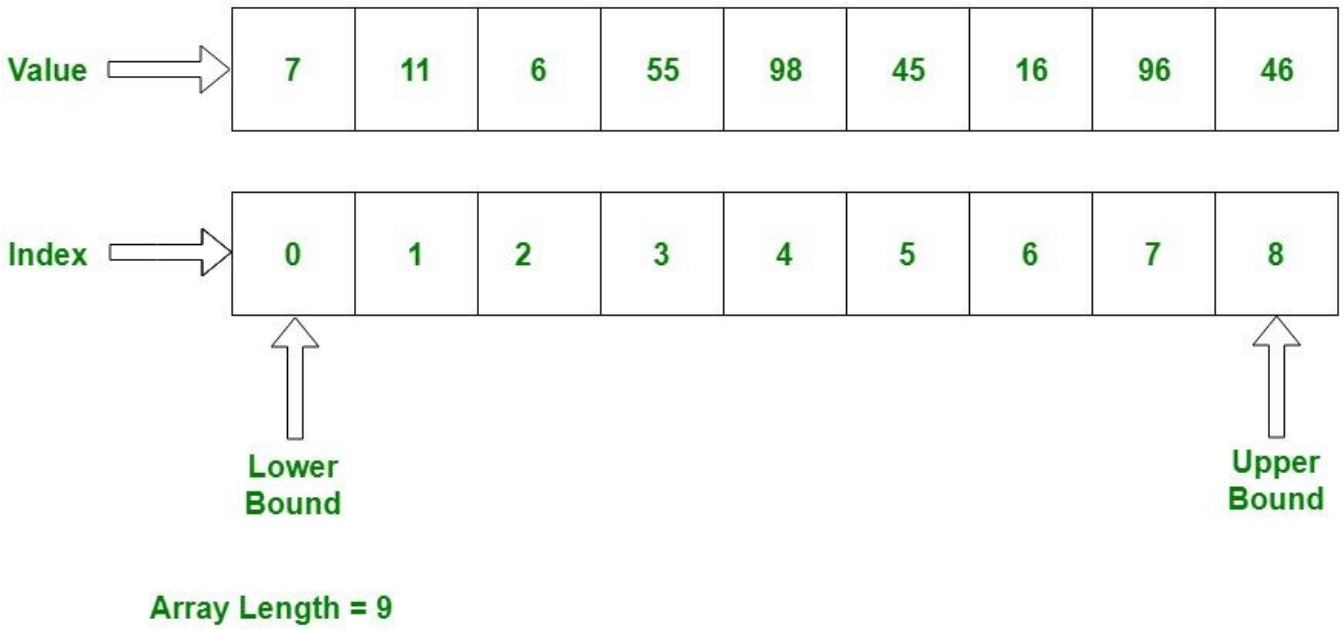
```
        confirm = Console.ReadLine();


    } while (confirm == "yes");
```

C# Arrays:

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.

OR

- An array is a collection of elements of a single data types stored in memory locations.
- Is a collection of related values placed in contiguous memory locations and these values are referenced using a common array name.



Example:

```
int[] myarray = new int[5];

myarray[0] = 66;
myarray[1] = 44;
myarray[2] = 11;
```

```
myarray[3] = 99;
myarray[4] = 88;


Console.WriteLine(myarray[0]);
Console.WriteLine(myarray[1]);
Console.WriteLine(myarray[2]);
Console.WriteLine(myarray[3]);
Console.WriteLine(myarray[4]);
```

Another Example:

```
int[] myarray = new int[5] {33,55,77,88,34};

Console.WriteLine(myarray[0]);
Console.WriteLine(myarray[1]);
Console.WriteLine(myarray[2]);
Console.WriteLine(myarray[3]);
Console.WriteLine(myarray[4]);
```

Another Example:

```
int[] myarray = {33,55,77,88,34};

Console.WriteLine(myarray[0]);
Console.WriteLine(myarray[1]);
Console.WriteLine(myarray[2]);
Console.WriteLine(myarray[3]);
Console.WriteLine(myarray[4]);
```

Another Example with string array:

```
Console.Write("Enter array Length : ");

int len = int.Parse(Console.ReadLine());

string[] names = new string[len];

for (int i = 0; i < len; i++)
{
```

```
    Console.Write("Enter " + (i + 1) + " Name : ");
    names[i] = Console.ReadLine();

}

Console.WriteLine("==================================\n\n");

for (int i = 0; i < names.Length; i++)
{
    Console.WriteLine((i + 1) + ") Name : " + names[i]);
}
```

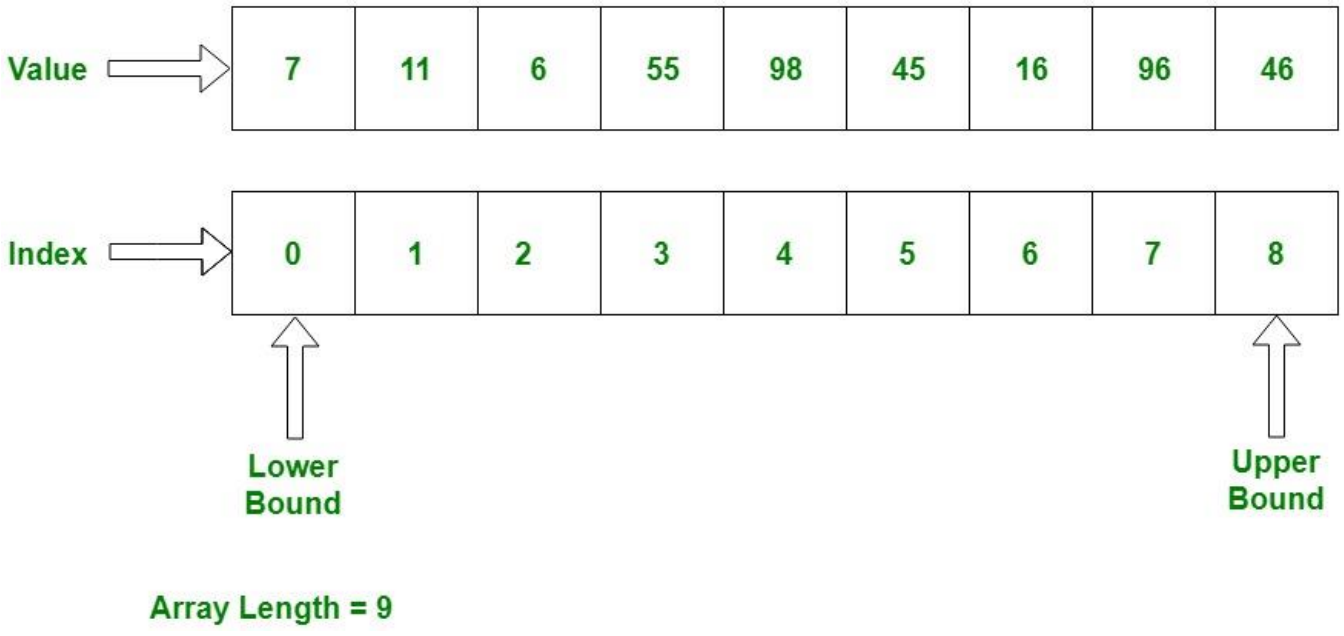
Types of Arrays:

There are two types of arrays.

1. Single dimensional array
2. Multi-dimensional array

Single Dimensional Array:

It is a simple array what I teach you up.



Multi-dimensional Array:

It is multi-dimensional array that is consist on rows and columns.

Example:

```
int[,] myarray = new int[2, 4];

myarray[0, 0] = 66;
myarray[0, 1] = 23;
myarray[0, 2] = 77;
myarray[0, 3] = 24;
myarray[1, 0] = 88;
myarray[1, 1] = 13;
myarray[1, 2] = 77;
myarray[1, 3] = 84;

Console.WriteLine(myarray[0,3]);
```

Another Example of 2D Array:

```
int[,] myarray = new int[2, 4]
{
   {56,67,78,11 },
   {77,23,99,57 }
};

Console.WriteLine(myarray[0,3]);
```

Another Example:

```
start:

   Console.Clear();

   Console.Write("Enter How many Row you require ?");

   int row = int.Parse(Console.ReadLine());

   Console.Write("Enter How many Column you require ?");

   int col = int.Parse(Console.ReadLine());
```

```
int[,] myarray = new int[row, col];

for (int i = 0; i < row; i++)
{

   for (int j = 0; j < col; j++)
   {
      Console.Write("Enter " + i + " " + j + " Number : ");
      myarray[i, j] = int.Parse(Console.ReadLine());

   }

}

Console.WriteLine("==============================");
Console.WriteLine("\n\n");

Console.WriteLine("----------------------");

for (int i = 0; i < row; i++)
{

   for (int j = 0; j < col; j++)
   {
      Console.Write(" | " + myarray[i, j] + "  |  ");
   }
   Console.WriteLine("");
}

Console.WriteLine("----------------------");

Console.WriteLine("DO you want to conitue  yes / no ? : ");

string op = Console.ReadLine();

if (op == "yes")
{
   goto start;
}
```

Assignment: Make ATM Machine with full options

Methods or Functions:

- It is piece of code to perform a specific work.
- Method and function is not different thing it is same thing.
- There are two stages of method: method declaration, method calling.

Method Types:

1) Static Methods
2) None Static Method / instance methods

Method has 4 types:

1. Take something return something
2. Take nothing return something
3. Take something return nothing
4. Take nothing return nothing

Syntax:

```
Access_modifier return_type method_name(parameter 1, parameter 2){
        Code of block
}
```

Example:

Method Declaration:

```
public void showmessage()
    {
       Console.WriteLine("This is Message line 1 ");
       Console.WriteLine("This is Message line 2 ");
       Console.WriteLine("This is Message line 3 ");
    }
```

Method Calling (In main method):

```
Program pa = new Program();

pa.showmessage();
```

Another Example (Parameterize Method):

```
public void showmessage(string name)
    {
      Console.WriteLine("Your Full Name : " + name);
    }
```

Calling Method:

```
Program pa = new Program();


      pa.showmessage("Farhan");
      pa.showmessage("Ali Raza");
      pa.showmessage("Ahmed Khan");
```

1) **Take Something return nothing Example:**

```
public void showmessage(string name)
    {
      Console.WriteLine("Your Full Name : " + name);
    }
```

2) **Take Something return Something Example:**

```
public int sum(int a , int b)
{
   int c = a + b;
   return c;
}
```

3) **Take Nothing Return Something Example:**

```
public int sum()
```

```
{
    int a = 20;
    int b = 30;

    int c = a + b;
    return c;
}
```

4) **Take Nothing Return Nothing Example:**

```
public void showmessage(string name)
{
    Console.WriteLine("Your Full Name : " + name);
}
```

Static Method (declaration):

```
public static void showmessage()
{
    Console.WriteLine("This is Message");
    Console.WriteLine("This is Message");
}
```

Static Method Calling:

```
showmessage();
```

Method Passing value and reference:

Pass by Value:

```
public static void passbyvalue(int a)
{
    a = a + 1;
    Console.WriteLine("Value in A = " + a);
}
```

Calling in Main Method:

```
    int value = 0;
```

```
passbyvalue(value);

Console.WriteLine(value);
```

Pass by Reference:

```
public static void passbyreference(ref int a)
    {
       a = a + 1;
       Console.WriteLine("Value in A = " + a);
    }
```

Calling in Main Method:

```
int value = 0;

passbyreference (ref value);

Console.WriteLine(value);
```

Pass by Out:

```
public static void passbyout(out int a)
    {
       a = 20;
       Console.WriteLine("Value in A = " + a);
    }
```

Calling in Main Method:

```
int value;

passbyout(out value);

Console.WriteLine(value);
```

var and dynamic Keyword:

Var Keyword:

- var keyword is used to store any type of data in its variable.
- Value of var variable is decided at compile time.
- GetType() method get the data type of any variable.
- Var variable cannot be used for property or return values from a function. They can only be used as local variable in a function.
- We cannot use var variable as a function parameter.

Example:

```
var a = 20;

Console.WriteLine(a);
```

Dynamic Keyword:

- Dynamic keyword is also used to store any type of data in its variable.
- Value of dynamic variable is decided at run time.
- Initialization is not mandatory when we declare a variable with dynamic keyword.
- Dynamic variables can be used to create properties and return values from a function.
- We can use dynamic variable as a function parameter.

Example:

```
dynamic a = "farhan";

a = 230;

Console.WriteLine(a);
```

Value Type:

A data type is a value type if it holds a data value within its own memory space. It means the variables of these data types directly contain values.

The following data types are all of value type:

- bool
- byte
- char
- decimal
- double
- enum
- float
- int
- long
- sbyte
- short
- struct
- uint
- ulong
- ushort

Reference Type:

Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data.