1. **What is .NET** — **Done**
2. **What is .NET Framework** — **Done**
3. **What is ASP.NET** — **Done**
   - 3.1. **ASP.NET web technologies** — **Done**
4. **What is Framework** — **Done**
5. **What is .NET Framework Architecture (CLR, BCL)** — **Done**
6. **Explain Execution of Program** — **Done**
7. **What is Visual Studio** — **Done**
8. **What is C# (Uses, Features)** — **Done**
9. **Basic Structure of C# Program** — **Done**
10. **Program (Print a "Hello World")** — **Done**
11. **Write() and WriteLine() method** — **Done**
12. **Read and ReadLine method** — **Done**
13. **What is Namespace** — **Done**
14. **What is Main Method** — **Done**
15. **What is Keywords** — **Done**
16. **What is Variable (Static, Non-static instance, Const, Readonly )** — **Done**
17. **What are Data Types** — **Done**
18. **Block** — **Done**
19. **Statement** — **Done**
20. **Expression** — **Done**
21. **Comments** — **Done**
22. **Indentation** — **Done**
23. **Data Type Conversion** — **Done**
24. **Operator** — **Done**
    - 24.1. **Arithmetic Operators**
    - 24.2. **Logical Operators**
    - 24.3. **Assignment Operators**
    - 24.4. **Relational Operators**
    - 24.5. **Arithmetic Programs**
25. **Selection Statement (Decision Making)** — **Done**
    - 25.1. **If**
    - 25.2. **If-else**
    - 25.3. **If-else-if**
    - 25.4. **Nested-if**
    - 25.5. **Switch Case**
26. **Control Flow Statement**
    - 26.1. **For Loop**
    - 26.2. **Nested For Loop**
    - 26.3. **While Loop**
    - 26.4. **Do-while Loop**
    - 26.5. **Foreach Loop**
    - 26.6. **Number Patterns**
    - 26.7. **Star Patterns**
27. **Jump Statement**
    - 27.1. **Break**
    - 27.2. **Continue**
    - 27.3. **Goto**
28. **Var and Dynamic Variable**
29. **Checked and Unchecked Keyword**
30. Boxing and Unboxing — https://www.youtube.com/watch?v=37MgjErWXeA&list=PLp_RsiLZjwQRVqZhQIddhU6b6k7bk5Tqc&index=74
31. Convert.ToString and ToString
32. Stack and Heap Memory / Value Type and Reference Type
33. Pass by Value and Pass by Reference

# .NET

**(Network Enabled Technology)**

## 1: .Net

.NET stands for **Network Enabled Technology is a free, open source, cross-platform, secure, reliable software platform** designed and developed by Microsoft, for building many **different types of modern, interactive, high performance, dynamic software applications** (like: Console Applications, Windows Desktop Applications (using Win forms), Web applications (using ASP.NET Web Forms, ASP.NET MVC, .NET Core), Mobile Applications (using Xamarin), Gaming applications (using Unity) that can be run on cross platforms.

## 2: .Net Framework

.Net Framework is a **free, open source, software development framework** designed and developed by Microsoft. It provides a **managed execution environment** along with a comprehensive **set of tools, libraries and modules** that developers can develop and run various types of applications.

**Managed code execution**: Offers memory management, garbage collection, and exception handling to ensure secure and efficient application execution.

- **First Beta Version**: 2000
- **First Version (.NET Framework 1.0):** Feb, 2002

## 3: ASP.NET

ASP.NET stands for **Active Server Page Network Enabled Technology** is a powerful **server-side web development framework** designed and developed by Microsoft, for building **modern, interactive, high performance, dynamic web applications**.

ASP.NET offers several frameworks for building web applications: Web Forms, ASP.NET MVC, ASP.NET Core (.NET), ASP.NET WebPages.

https://dotnettutorials.net/lesson/overview-of-microsoft-web-technologies/

**Microsoft Web Technologies**

- ASP.NET Web Forms (.aspx)
- ASP.NET MVC
- ASP.NET Razor Pages
- ASP.NET Web API
- .NET Core (.NET 5→ .NET 6→ .NET 7→ .NET 8→ .NET 9)

**1: Win Forms:** Win Forms (**Windows Form Application**) is a **GUI based class library of .NET Framework** used for **building desktop applications**. Win Forms was introduced with the first version of the .NET Framework in **2002**, but it is used exclusively for creating **Windows desktop applications**.

**2: Web Forms:** ASP.NET Web Forms is a **traditional web development framework** designed and developed by Microsoft used for **building dynamic web applications** using a drag-and-drop, event-driven model. Web Forms was introduced in the .NET Framework 1.0 version, released in 2002.

**3: ASP.NET MVC:** ASP.NET MVC stands for **Active Server Page Network Enabled Technology Model-View-Controller is a web development framework** designed and developed by Microsoft that adopts the **Model-View-Controller architecture pattern** for **building modern, interactive, high performance, dynamic web applications.** This pattern helps to achieve separation of concerns. Here the entire application is divided into three main components. They are Models, Views, and Controllers. The **model contains the business logic**, the **View manages the user interface logic**, and the **Controller is a mediator, handles the user input logic**. It was introduced in the .NET Framework 3.5 version, which was released in 2009.

MVC Course: https://dotnettutorials.net/course/asp-dot-net-mvc-tutorials/

MVC Lesson: https://dotnettutorials.net/lesson/asp-dot-net-mvc-architecture/

**4: ASP.NET Core:** ASP.NET Core (.NET) is an updated version of ASP.NET. It is a **free, open-source, modular (separation of concerns), cross platform, high performance and cloud optimized web development framework** based on **Model-View-Controller architecture pattern** created by Microsoft for building **modern, interactive, high performance, dynamic, cloud-enabled web applications**. It offers improved **performance, scalability, and modularity compared to its predecessor, ASP.NET**. The initial version of .NET Core was released in 2016. It is not a continuous part of the ASP.NET 4.x Framework.

**5: ASP.NET Web Pages:** ASP.NET Web Pages is a **lightweight web development framework** that allows developers to combine HTML, CSS, and server-side code (using Razor Pages) in the same file. ASP.NET Web Pages was first introduced with **ASP.NET Web Matrix** in **2010.**

Course: https://dotnettutorials.net/course/asp-net-core-tutorials/

Lesson: https://dotnettutorials.net/lesson/introduction-asp-net-core-mvc/

**4: Framework**: Framework is a **structured collection of pre-written code** that refers to a **set of tools and modules** that can be used to build software applications.

**5: .NET Framework Architecture:** The .NET Framework architecture consists of two primary components:

1. **Common Language Runtime (CLR)**
2. **Base Class Library (BCL)**

## 5.1: Common Language Runtime (CLR)

CLR stands for **Common Language Runtime** is a **virtual managed execution engine responsible for managing the execution of .NET applications programs.** It provides a multi-language execution environment. The CLR manages various tasks, including **thread management, garbage collection, type-safety, exception handling etc**.

**Components of CLR:**

1. **CLS (Common Language Specification)**
2. **CTS (Common Type System)**
3. **GC (Garbage Collector)**
   - **Workstation garbage collector**
   - **Server garbage collector**
4. **JIT (Just-In-Time Compiler)**
5. **Memory Management**
6. **Exception Manager**
7. **Security Manager**
   - **CAS (Code Access Security)**
   - **CV (Code Verification)**

- **CLS (Common Language Specification):** CLS stands for **common language specification** that is responsible for the conversion of different programming language syntactic rules and regulations into CLR compatible format.
- **CTS (Common Type System):** CTS stands for **common type system**, every programming language has its own data type system, so CTS is responsible for **understanding all the data type systems of different programming languages** and converting them into CLR compatible format.
- **GC (Garbage Collector):** GC stands for **Garbage Collector**. Garbage Collector automates **memory management** and is a process of **freeing up memory** that is captured by unwanted objects and ensuring efficient memory utilization.
- **JIT (Just In Time Compiler):** JIT stands for **Just-In-Time Compiler** and **converts MSIL into native machine code** and this native code is understandable by the OS.
- **Memory Manager:** The Memory Manager **allocates the memory for the variables and objects** required during the application's execution.
- **Exception Manager:** The Exception Manager **handles the runtime exceptions** by **control to execute the catch and finally blocks**.

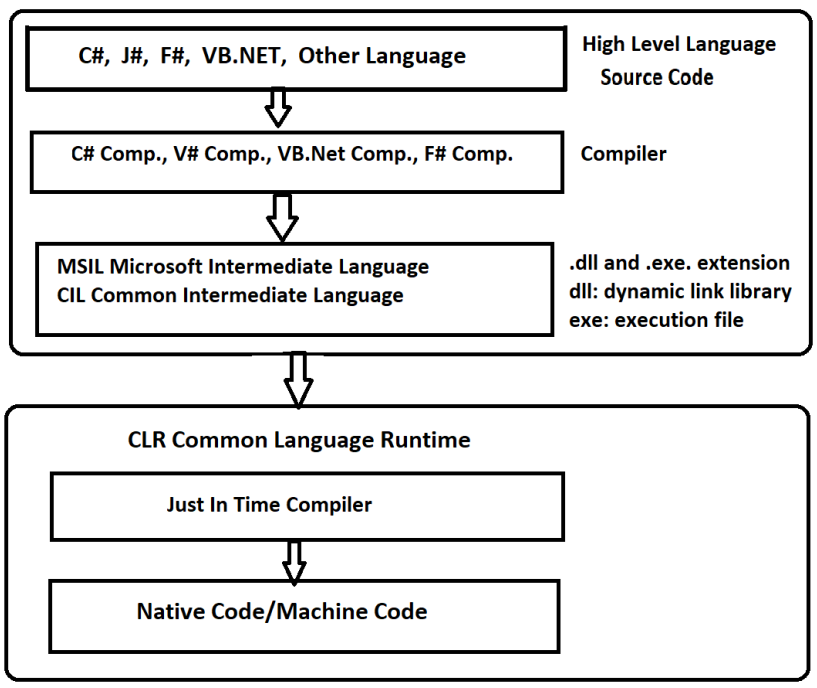
## 5.2: Base Class Library (BCL)

The Base class library is a **collection of pre-built classes, methods, and interfaces that provide core functionality** to .NET applications. These libraries enable developers to create applications more efficiently. Base Class Library is also known as **Framework Class Library.**

- **Data Types:** Predefined data types such as integers, strings, and collections.
- **Input /Output Operations:** Tools for reading/writing data streams.
- **Networking:** Libraries for communication over networks.
- **File Handling:** Methods for interacting with file systems.
- **Threading:** Supports multi-threaded application development.
- **Date and Time:** Provides classes to handle date and time operations.
- **Collections and Generics:** Offers classes to manage collections of data.

## 6: Execution of Program/Execution Process

- .Net applications are typically written in C# programming language.
- Source code is first compiled into **Common Intermediate Language (CIL) also called MSIL (Microsoft Intermediate Language)** using a **specific language compiler**. This code is platform independent.
- **Note: Intermediate Language is also called Managed Code.**
- Compiled code is stored in assemblies files have extension of .dll (dynamic link library) or .exe file extension. If **we compile Windows and Console Application** → we get assembly of the **.exe extension**. Whereas when **we compile a Web Project**, we get an assembly of type **.dll extension**.
- When an app runs, the CLR takes over these assemblies and uses Just-In-Time Compiler to turn it into machine code.




## 7: Visual Studio:

Visual Studio is a **Powerful and Comprehensive Integrated Development Environment (IDE)** developed by Microsoft that supports code design, development, test, compiling, debugging, Interface design, Server management and performance analysis and deploy application efficiently.

## 8: C# (C-Sharp)

C# is a **modern, general-purpose, open sources, cross-platform, server-side, fully object-oriented programming language** developed by Microsoft. It features a **robust type system and an advanced exception-handling mechanism**. C# is designed to build a variety of software applications that can run across platforms such as Windows, Linux, and macOS. Anders Hejlsberg, a prominent computer scientist, is credited as the creator of C#

### 8.1 Uses:

- Console Applications
- Window Applications
- Web Applications
- Mobile Applications
- Gaming Applications

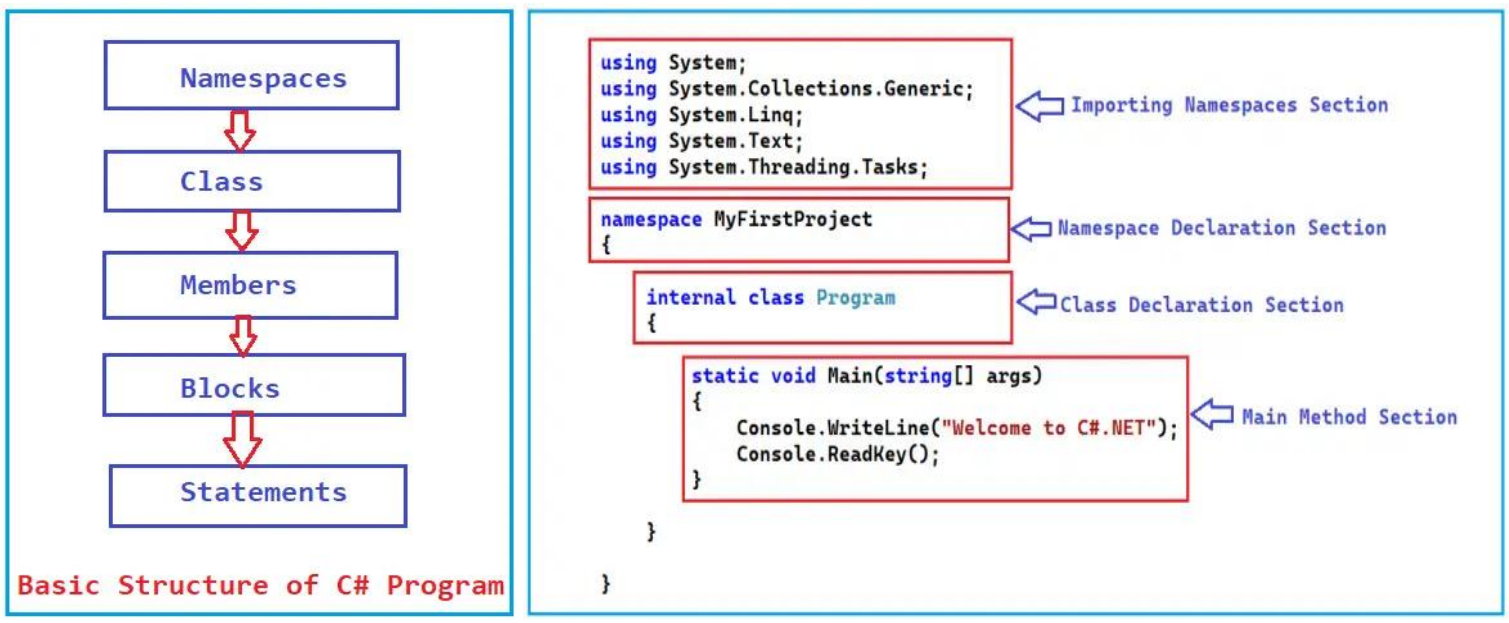
### 8.2 Version of C#:

- Current Version: 12.0 (Released in November 2023)

### 8.3 Features of C#:

- Modern General Purpose Server Side Object Oriented Programming Language
- Case Sensitive Language
- Interoperability Language (Different programming languages to interact within the same system)
- Rich Library (C# provides a lot of inbuilt functions that make development fast)
- Provides automatic garbage collector facility
- Strong Exception Handling
- Support Operator overloading, Method hiding
- Supports goto method
- Extension : .cs

### 9. Basic Structure of C# Program




### 10. Program (Print a “Hello World”)

```
using System;                          //using keywords a predefined keyword, “using” is used to import other namespace/ assemblies.
                                       //System is a namespace.

namespace FirstProgram                 //Namespace is a collection of classes and used to organize too many classes.
{
  class Program                        //Class is a predefined keyword used to define class.
                                       //Program is Class name.
 {
   static void Main(string[] args)     //static is a predefined keyword,
                                       //void is a return type, does not return any value,
                                       //Main method is an entry point of application, used to execute and call methods.
   {
     Console.WriteLine("Hello World"); //Console is a class, defined in System (namespace).,
                                       //WriteLine is a method defined in Console Class and used to print output and moves to the next line after the print the output
     Console.ReadLine();               //ReadLine is a method defined in Console Class used to hold screen and read all the characters from the input stream.
   }
  }
```

Output: Hello World

### 10.1 Program (Print your Name)

```
internal class Program
 {
    static void Main(string[] args)
    {
       Console.WriteLine("Your First Name");
       string firstname = Console.ReadLine();
       Console.WriteLine("Your Last Name");
       string lastname = Console.ReadLine();
       Console.WriteLine("Your full name: " + firstname +" "+ lastname);
       Console.ReadLine();
    }
 }
```

Output: Enter your First name:
Developer
Enter your Last Name:
Sharma
Your full name is: Developer Sharma

## 11. Write() and WriteLine() method

Write and WriteLine are methods of the **System.Console class.**

**The Write method** used to only print the value.

**The WriteLine method** prints the value and automatically moves the cursor to the next line after the print the output.

```
Console.Write("Hello");
Console.Write("World");
Output: Hello World
```

```
Console.WriteLine("Hello");
Console.WriteLine("World");
Output: Hello
        World
```

## 12. Read and ReadLine method

Read and ReadLine are methods of **System.IO.TextReader class.**

**The Read Method** used to read a single character from the input stream.  Return type int

```
{
int ch = Console.Read();                // Reads a single character
}
```

**The ReadLine Method** is used to read all the characters from the input stream. Return type string

```
{
string input = Console.ReadLine();      // Reads a line of text
}
```

## 13: Namespace

Namespace is a **collection of classes** and **used to organize too many Classes** so that it can be easy to handle the application.

Namespace is a container that can contain other namespaces.

Organize: **Organize** code into a structured format.

## 14: Main Method

The main method is an entry point of application. It is used to execute and call other methods.

## 15: Keywords

Keywords are **predefined reserved words in the programming language** that have special meaning in a program. The meaning of keywords can't be changed. C# has a total of 79 keywords. All these keywords are in lowercase.

| | |
|---|---|
| **Data Types:** | sbyte, byte, short, ushort, int, uint, long, ulong, float, double, decimal, char, string, bool |
| **Boolean Values:** | true, false |
| **Control Flow & Logic:** | if, else, switch, case, break, continue, goto, for, while, do-while, foreach |
| **Access Modifiers:** | public, private, protected, internal |
| **OOP:** | class, static, abstract, interface, new, virtual, override, sealed, readonly, ref, const, this, base, params |
| **Exception Handling:** | try, catch, finally, throw |
| **Other Keywords:** | namespace, using, void, return, delegate, enum, implicit, explicit, default, is, as, typeof, checked, unchecked, var, readonly, extern, unsafe, fixed, async, await, lock |

## 16. Variable

Variable is just a name used to store data or values.

- Variable names are case sensitive.
- Variable names can start from the alphabet and underscore.
- There can be a combination of alpha and numbers.
- There will be no space between variable names.
- There will be no special character.
- Variables can't start from numbers. Variables can start from numbers.
- We cannot use reserved keyword as variable name
- **Example 1:** int num; int is a data type, num is a variable.
- **Example 2:** int num = 20 int is a data type, num is a variable, initializer to 20.

Types of variables:

- **Static Variable**
- **Non-static instance Variable**
- **Const Variable**
- **Readonly Variable**

## 16.1 Static Variable

Declare a variable using a **static** keyword.
If we declare a variable inside a **static block** then also that variable is a static variable.
If we declare a variable inside the **main method** then that variable will be a static variable.
**Example 1:** static int counter = 0;

## 16.2 Non-Static Variable

When a variable declared within a class.

## 16.3 Constant Variable

Declare a variable by using the **'const'** keyword.
Value **assigned at the time of declaration** and cannot be changed.

- We have to initialize a constant variable at the time of declaration, else it will give the error.
- Constants variables are value types.
- Used the **"Const"** keyword for declaration.

**Example 1:** const double PI = 3.14;
**Example 2:** public const string company_name= "xyz_ company";

```
internal class Program
{
   static int x = 100;                          // static variable
   int y = 200;                                 // by default non-static private variable
   const float PI = 3.14f;                      // const variable

   static void Main(string[] args)
   {
      int a = 12;                               //it is a static variable, we have to initialize this, by default it is a static variable.
      Console.WriteLine($"a value: {a}");
      Console.WriteLine($"x value : {x}");
      Program program = new Program();
      Console.WriteLine(program.y);
      Console.WriteLine(PI);

      Console.ReadLine();
   }
}
```

**Output**:
a value: 12
x value : 100
y value : 200
PI value: 3.14

## 16.3 Readonly Variable

A **readonly variable** can be initialized only at the time of declaration or inside the constructor. Once initialized, it cannot be modified.
**Example 1:** public readonly int id = 1001;

**17 Data Type:** Data Type is used to **define the type of the value**. It defines **which type of value we are using and stores in a variable**. There are 7 types of Data types in C#.

**1: Integer:** Used to store numeric values.

- Signed Integer (takes positive and negative value both.)
- Unsigned Integer (takes only positive value)

**Signed Integer**

| | | |
|---|---|---|
| sbyte | (-128 to 127) | (1 byte) |
| short | (-32,768 to 32,767) | (2 byte) |
| int | (-2,147,438,647 to 2,147,438,648) | (4 byte) |
| long | | (8 byte) |

**Unsigned Integer**

| | | |
|---|---|---|
| byte | (0 to 255) | (1 byte) |
| ushort | (0 to 65535) | (2 byte) |
| uint | (0 to 4,294,967,295) | (4 byte) |
| ulong | | (8 byte) |

**2: Boolean: (1 byte):** Bool Keyword is used for Boolean data type which only stores TRUE and FALSE.

```
int a=20, b=30;
bool c=a<b;
Console.WriteLine(c);
Output: TRUE
```

**3: Float: (4 byte),**
**Precision: 7 digits only. Use: Single**

```
float num1 = 23.34263f;
```

**4: Double: (8 byte),**
**Precision: 15-16 digits only Use: Double**

```
double num2 = 234.2425325d;
```

**5: Decimal: (16 byte),**
**Precision: 28-29 digit Use: Decimal**

```
decimal num3= 32497948.324827384829m;
```

**6: String:** String is a collection of characters put in double quotes. A string is a sequence of Characters.

```
string str1=”Hello World”;
```

**7: Char: (2 byte)**
Char used to store only one character put in a single quote.

```
Char ch1=’D’;
```

**Note: Escape sequence:** (\n)= new line), (\t=horizontal tab)
**Verbatim sequence:** @ (use before string), More Readable

```
static void Main(string[] args)
{
    sbyte num1 = 56;
    byte num2 = 158;
    short num3 = 12345;
    ushort num4 = 53432;
    int num5 = 938723212;
    uint num6 = 234321342;
    long num7 = 9873829922383;
    ulong num8 = 9876545672289;
    float num9 = 83372.48f;
    double num10 = 876783.3487653d;
    decimal num11 = 98878976.45432345m;
    char ch = 'R';
    string str = "Hello World";

    Console.WriteLine(num1);
    Console.WriteLine(num2);
    Console.WriteLine(num3);
    Console.WriteLine(num4);
    Console.WriteLine(num5);
    Console.WriteLine(num6);
    Console.WriteLine(num7);
    Console.WriteLine(num8);
    Console.WriteLine(num9);
    Console.WriteLine(num10);
    Console.WriteLine(num11);
    Console.WriteLine(ch);
    Console.WriteLine(str);
    Console.ReadLine();
}
```

Output:

```
56
158
12345
53432
938723212
234321342
9873829922383
9876545672289
83372.48
876783.3487653
98878976.45432345
R
Hello World
```

## Min and Max Range of Data Type

```
Console.WriteLine(sbyte.MinValue +" "+ "to"+" "+ sbyte.MaxValue);    //min and max range of datatype
```

## Taking User Input (Data type)

```
int num; / short num
Console.WriteLine("Enter the Num ");                    //2 byte= 16 bit, take=5 digit
num=Convert.ToInt16(Console.ReadLine());                //range= (-32768 to 32767)
Console.WriteLine("Your num is: " + num);
Console.ReadLine();
-----------------------------------------------------------------
int num;
Console.WriteLine("Enter the Num");                     //4 byte= 32 bit, take= 10 digit
num=Convert.ToInt32(Console.ReadLine());                //range= (-2,147,483,648 to 2,147,483,647)
Console.WriteLine("Your num is: " + num);
Console.ReadLine();
-----------------------------------------------------------------
long num;                                               //8 byte = 64 bit
Console.WriteLine("Enter the Num");                     //range= (-9223372036854775808 to 9223372036854775807)
num=Convert.ToInt64(Console.ReadLine());
Console.WriteLine("Your num is: " + num)
Console.ReadLine();
-----------------------------------------------------------------
byte age;                                               //1 byte=8 bit
Console.WriteLine("Enter the Num");                     //range= 0 to 255
age=Convert.ToByte(Console.ReadLine());
Console.WriteLine("Your num is: " +age);
Console.ReadLine();
-----------------------------------------------------------------
float num;                                              //4 Byte= 32 bit
Console.WriteLine("Enter the Num");
num=Convert.ToSingle(Console.ReadLine());
Console.WriteLine("Your num is: " + num);
Console.ReadLine();
-----------------------------------------------------------------
double num;                                             //8 Byte= 64 bit
Console.WriteLine("Enter the Num");
num=Convert.ToDouble(Console.ReadLine());
Console.WriteLine("Your num is: " + num);
-----------------------------------------------------------------
decimal num;                                            //16 Byte=  128 bit
Console.WriteLine("Enter the Num");
num=Convert.ToDecimal(Console.ReadLine());
Console.WriteLine("Your num is: " + num);
}}
-----------------------------------------------------------------
char chr;                                               //only one character , 1 byte =8 bit
Console.WriteLine("Enter the Choice_Char");
ch =Convert.ToChar(Console.ReadLine());
Console.WriteLine("Your num is: " + ch);
Console.ReadLine();
-----------------------------------------------------------------
string str;
Console.WriteLine("Enter the Choice_String");
str = Console.ReadLine();
Console.WriteLine("Your num is: " + str);
Console.ReadLine();
-----------------------------------------------------------------
```

## 18. BLOCK

- A block is a **grouping of multiple statements** or block is a **code segment enclosed in curly braces**.

## 19. STATEMENT

- To perform a task, programmers provide instructions, these instructions are called statements.
- Statements are referred to as logical grouping of: Variable, Data Type, Operators, Constants Variable, Keywords, Escape Sequence Characters that perform a specific task.
- In C#, a statement ends with a semicolon(;).
- The program contains multiple statements.
- Statement is used to logical flow in the program: Initialize a variable, take the input, call a method, display the output.

  **Types of statement:**

  1. Selection Statement(If-else, switch)
  2. Control (Iteration) Statement(Loops)
  3. Jump Statement (Break, Continue)
  4. Exception Handling

## 20. EXPRESSION

- An Expression is a combination of Operators and Operands.
- Ends with semicolon (;).
- Example: int c= a+c;

## 21. COMMENTS

- Comments are used to explain or give more information about the code.
- Restrict the code execution.
- Provide an overview about the project.

## 22. INDENTATION

- Indentation refers to the white spaces at the beginning of the code line.
- More Readability.
- Code is clean.

## 23. Data Type Conversion/ Type Casting

The process of **converting one data type to another data type** is known as Type Casting.
There are 2 types of conversion:

**1: Implicit Type Conversion:** Implicit Type Casting done by the compiler. Smaller data types are converted to larger data types. Hence, there is no loss of data during the conversion.

| Convert from Data Type | Convert to Data Type |
|---|---|
| byte | short, int, long, float, double |
| short | int, long, float, double |
| int | long, float, double |
| long | float, double |
| float | double |

```
int a = 20;
float b = Convert.ToSingle(a);                  //implicit type conversion
double c = Convert.ToDouble(a);                 //implicit type conversion
decimal d = Convert.ToDecimal(a);               //implicit type conversion
Console.WriteLine("answer of b: " + b);
Console.WriteLine("answer of b: " + c);
Console.WriteLine("answer of b: " + d);
```

**Output: 20**
**20**
**20**

or

```
string num = "20";
int a = Convert.ToInt32(num);                   //implicit type conversion
float b = Convert.ToSingle(num);                //implicit type conversion
double c = Convert.ToDouble(num);               //implicit type conversion
decimal d = Convert.ToDecimal(num);             //implicit type conversion
Console.WriteLine("answer of b: " + a);
Console.WriteLine("answer of b: " + b);
Console.WriteLine("answer of b: " + c);
Console.WriteLine("answer of b: " + d);
```

**Output: 20**
**20**
**20**
**20**

**2: Explicit Type Conversion:** Explicit Type Casting done by the user. If we want a large data type to a small data type then we will need explicit type casting. Hence, there can be loss of data during conversion. **Example:** float to int, but loss of data.

```
float a = 20.478f;
int b = Convert.ToInt32(a);                 //explicit type conversion
Console.WriteLine("answer of b: " + b);
Console.ReadLine();Output: 21               //(here there will be data lost), here int is a cast expression, that explicitly converts float type into int
```

Output: 20

```
double a = 20.478d;
int b = Convert.ToInt32(a);                 //explicit type conversion
Console.WriteLine("answer of b: " + b);
Console.ReadLine();                         //(here there will be data lost), here int is a cast expression, that explicitly converts double type into int
```

## 24. OPERATOR

- Operator is used to perform an operation on operands.
- Example: 2+2= 4
- Values are called **Operands** and Symbols are called **Operators**.

There are 3 types of operator:

- Binary Operator
  - Arithmetic Operator
  - Relational Operator
  - Logical Operator
  - Assignment Operator
- Unary Operator (++, --) work with one operand, Increment, Decrement
- Ternary Operator (?)

1: **Arithmetic Operator**: Arithmetic Operators are used to perform numeric calculations. (+,-, *,/,%)
2: **Relational Operator (Comparison Operator):** Answer will be in TRUE and FALSE. (==, !=, >,<,>=,<=)

```
int a = 40, b = 20;                                  //predefined values
int c = a + b;
int d = a - b;
int e = a * b;
int f = a / b;
int g = a % b;
Console.WriteLine("answer of c :" + c);          60
Console.WriteLine("answer of d :" + d);          20
Console.WriteLine("answer of e :" + e);          800
Console.WriteLine("answer of f :" + f);          2
Console.WriteLine("answer of g :" + g);          0
Console.ReadLine();
```

**or**

```
static void Main(string[] args)
 {
     //arithmetic operators
     float a, b, c, d, e, f, g;
     Console.WriteLine("enter the first value");    //User input
     a = Convert.ToSingle(Console.ReadLine());
     Console.WriteLine("enter the second value");
     b = Convert.ToSingle(Console.ReadLine());
     c = a + b;
     d = a - b;
     e = a * b;
     f = a / b;
     g = a % b;
     Console.WriteLine("answer of c :" + c);
     Console.WriteLine("answer of d :" + d);
     Console.WriteLine("answer of e :" + e);
     Console.WriteLine("answer of f :" + f);
     Console.WriteLine("answer of g :" + g);
     Console.WriteLine("--------------------");

     //relational operators
     bool k = a > b;
     bool l = a < b;
     bool m = a == b;
     bool n = a != b;
     bool o = a >= b;
     bool p = a <= b;
     Console.WriteLine("answer of k :" + k);
     Console.WriteLine("answer of l :" + l);
     Console.WriteLine("answer of m :" + m);
     Console.WriteLine("answer of n :" + n);
     Console.WriteLine("answer of o :" + o);
     Console.WriteLine("answer of p :" + p);
     Console.ReadLine();
 }
enter the first value
40
enter the second value
20
answer of c :60
answer of d :20
answer of e :800
answer of f :2
answer of g :0
--------------------
answer of k :True
answer of l :False
answer of m :False
answer of n :True
answer of o :True
answer of p :False
```

3: **Logical Operator (Conditional Operator):** Answer will be in TRUE and FALSE

AND (&&) =return true is both value true

0 0 0
0 1 0
1 0 0
1 1 1

OR(||)= return true if one value is true

0 0 0
0 1 1
1 0 1
1 1 1

```
Console.WriteLine("enter the value 1");
int a = int.Parse(Console.ReadLine());          //user input
Console.WriteLine("enter the value 2");
int b = int.Parse(Console.ReadLine());
bool c = (a < b) && (a <= b);  / bool c = (a < b) || (a <= b);
Console.WriteLine("The answer is :" + c);
Console.ReadLine();
Output
enter the value 1: 20
enter the value 2: 30
The answer is: True
```

**4: Assignment Operator:** Assignment operator is used to **assign the value** of the right side of the operand. (=, +=, -= , *=,  /=)

```
int a = 5;
a += 1;
Console.WriteLine(a);
Console.ReadLine();
```

Output: 6

**5: Increment and Decrement Operator:** Unary Operator (++,--)

- Increment operator (++) is used to increase the value by 1.
- Decrement operator (--) is used to decrease the value by 1.
- Used with one operand.
- If the operator is placed before the operand called “pre-increment” or “pre-decrement”.
- If the operator is placed after the operand called “post-increment” or “post-decrement”.

```
int a = 5;
Console.WriteLine(++a);       //pre-increment
Console.ReadLine();
Output: 6

int a = 5;
Console.WriteLine(a++);       //post-increment
Console.WriteLine(a);
Console.ReadLine();
Output: 5
        6
```

## 25. Selection Statement (Decision Making)

- Selection statements are **used to control the flow** of the program.
- Executes a particular block based on a condition.
- Selection statements are referred to as Decision Making constructs.
- Types of Selection Statements
    1. If,
    2. If-else,
    3. If–else if-else, (calculator)
    4. Nested –if,
    5. Switch Case

**1: If Statement-** If the condition evaluates to True, then if statement is executed.

```
int a, b;
Console.WriteLine("enter the value of a: ");
a = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("enter the value of a: ");
b = Convert.ToInt32(Console.ReadLine());
if (a > b)
{
    Console.WriteLine("a is greater");
}
```

**2: If-Else Statement** –If 'if' statement is true, then **if statement will execute otherwise else statement will execute**.

```
int a, b;
Console.WriteLine("enter the value of a: ");
a = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("enter the value of a: ");
b = Convert.ToInt32(Console.ReadLine());
if (a > b)
{
    Console.WriteLine("a is greater");
}
else
{
    Console.WriteLine("b is greater");
}
```

**3: If - else if - else**: Allows you to check multiple conditions.

```
int a, b, c;
Console.WriteLine("enter the value of a: ");
a = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("enter the value of b: ");
b = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("enter the value of c: ");
c = Convert.ToInt32(Console.ReadLine());
if (a > b && a > c)
{
    Console.WriteLine("a is greater");
}
else if (b > c && b > a)
{
    Console.WriteLine("b is greate");
}
else
{
    Console.WriteLine("c is greate");
}
```

**4: Nested If:** If condition **contains another if condition.**

```
int a, b, c;
Console.WriteLine("enter the value of a: ");
a = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("enter the value of b: ");
b = Convert.ToInt32(Console.ReadLine());

if (a > b)
{
    if (a == 10)
    {
        Console.WriteLine("a is 10");
    }
    else
    {
        Console.WriteLine("a is greater");
    }
}
else
{
    Console.WriteLine("b is greater");
}
```

**5: Switch Case** - The switch case executes one statement from multiple conditions. Code is clean and readable.

```
 int week_number;
 Console.WriteLine("Enter the week_number");                    //taking user input
 week_number = Convert.ToInt16(Console.ReadLine());
 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;
     case 5:
         Console.WriteLine("Friday");
         break;
     case 6:
         Console.WriteLine("Saturday");
         break;
     case 7:
         Console.WriteLine("Sunday");
         break;
     default:
         Console.WriteLine("Invalid week_number");
         break;
 }
```

Or

```
string name;
 Console.WriteLine("Enter name");
 name = Console.ReadLine();
 switch (name)
 {
     case "Sudarshan":
         Console.WriteLine("Welcome Sudarshan");
         break;
     case "Ajay":
         Console.WriteLine("Welcome Ajay");
         break;
     default:
         Console.WriteLine("Invalid name");
         break;
 }
```

Or

```
char ch;
Console.WriteLine(“Enter an alphabet”);
ch = Convert.ToChar(Console.ReadLine());
switch (Char.ToLower(ch))
{
case ‘a’:
case ‘e’:
case ‘i’:
case ‘o’:
case ‘u’:
Console.WriteLine(“Vowel”);
break;
default:
Console.WriteLine(“Not a vowel”);
break;
```

## 26. Control Flow Statement

**Loop:** Looping is a process of **repeating a block of code multiple times based on a condition**.

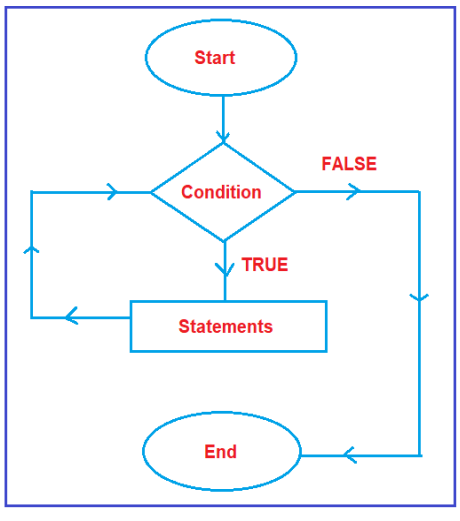
Control statements are used to execute a block of code no of times.

- For Loop
- Nested For Loop
- While Loop
- Do-While Loop
- Foreach Loop

**1: For Loop**: For loop is used to **iterate a part** of the program. For Loop is known as Counter Loop.

**Note:** If the number of iterations is fixed, it is recommended to use for loop.

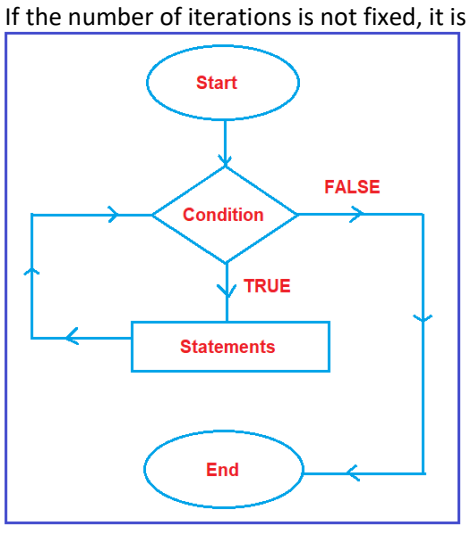
**For Loop has 4 statements: (Initialization, Condition, Execution, Increment/decrement)**




**2: Nested for Loop:** Nested For Loop consists of multiple for loop statements. When for loop is enclosed inside another for loop.

**3: While loop:** While loop is used to iterate a part of the program repeatedly as long as condition is true. When the condition is false it will terminate the loop execution.

**Note:** If the number of iterations is not fixed, it is recommended to use while loop rather than for loop.



### 4: Do-While Loop:

The do-while loop is similar to while loop, however it is always execute at least once without the condition being checked.

```
int i = 1;
do
{
Console.WriteLine(i);
i++;
}
while (i < 0);
Console.WriteLine("Loop terminates");
Console.ReadLine();
```

**Output:** 1

### 5: Foreach Loop:

Foreach is an extension of for loop. It is used to perform actions on large data collections. Reads every element in the specified array.
Allows to execute a block of code in the array.

## 27. Jump Statements

Jump statements are used to transfer control from one point to another.
In C# There are 3 types of jump statements.
**Break, Continue, Goto**

**1: Break Statement:** Break Statement stops the entire process of the loop and terminates.

```
for (int i = 1; i <= 10; i++)
{
    if (i == 6)
    {
        break;
    }
    Console.WriteLine(i);
}
Console.ReadLine();
```

Output:1 2 3 4 5

**2: Continue Statement:** Continue Statement stops the current iteration of the loop. It doesn't terminate.

```
for (int i = 1; i <= 10; i++)
{
    if (i == 6)
    {
      continue;
    }
    Console.WriteLine(i);
}
Console.ReadLine();
```

Output: 1 2 3 4 5 7 8 9 10

---

```
for (int i=1; i<=10;i++)
{
if (i % 2 == 0) / (i % 2 == 1)
{ continue; }
Console.WriteLine("Odd Num :" + i);
```

Output:
Odd num : 1 3 5 7 9,  Even Num: 2 4 6 8 10

---

### 3: Goto Statement:

```
for (int i=1; i<=10;i++)
{
if (i == 5)
{
Goto stop;
}
Console.WriteLine(i);
}
Console.writeLine("Loop terminates");

Stop:
Console.writeLine("Exit");
```

Output:
1
2
3
4
5

## 28. Var and Dynamic Variable

### Var variable

1. Var keyword is **used to store any type of data**.
2. Example: var a =20;
   var b=”Hello”;
3. The value of the **var variable is decided at compile time**.
4. We have to **initialize the variable** with var keyword.
5. If we want to check the type of value, use **GetType().**
6. Var variable is a **value** type.
7. Var variable **cannot be used for property or return values from a function.**
8. Var variable **can’t be used as function parameter**.

   **Example:**
   ```
   public static var show()          return value      it will show compile time error
   {
      return
   }
   ```
   **Example:**
   ```
   public static int show (var a, var b)    function parameter   it will show compile time error
   {
      return a+b;
   }
   ```
9. When we initialize the var variable with some value then we can’t change the value of var variable.
10. **Intellisense is available for var types of variables**. because it is compile time.
11. Example:
```
var b=”Hello”
Console.WriteLine(b);
Console.WriteLine(b.Length);
Console.WriteLine(b.GetType());
```

### Dynamic Variable

1. A dynamic variable is also **used to store any type of data**.
2. Example: dynamic a =20;  here we can use the same variable name with different values.
   a=”Hello”;
3. The Value of the **dynamic variable is decided at Run time**.
4. Initializing is not mandatory when we declare a dynamic variable. **Example: dynamic a;**
5. If we want to check the type of value, use **GetType().**
6. Dynamic variable is **reference** type.
7. Dynamic Variables can be used to create properties and return values from a function.
8. Dynamic variables can be used as a function parameter.
9. When we initialize the dynamic variable with some value then we can change the value of the dynamic variable.
10. **Intellisense is not available for dynamic types of variables** until execution.
11. Example:
```
Dynamic a=10;
a=”Hello”;
Console.WriteLine(a);
Console.WriteLine(a.GetType());
```
Output: System.string

**Or**

```
Public static void Show(dynamic a)
{
  Console.WriteLine(a);
}

Static main method
{
program.Show(5);
program.Show(“ABC”);
program.Show(‘A’);
program.Show(true);
}
```

## 29. Checked and Unchecked Keyword

Checked and unchecked keywords are **used to control integral type overflow exceptions**.

Example 1:

```
int a = 2147483647;
int b = 2147483647;
int c = a + b;
Console.WriteLine(c);
```

Output: -2
No exception will be thrown.

**Checked:** The checked keyword is used to explicitly **check overflow** and conversion of integral type values at compile time.
**Note:** It is used to ensure that the left hand side data type is not overflowed.

```
int a = 2147483647;
int b = 2147483647;
int c = checked(a + b);
Console.WriteLine(c);
```

Exception: System.OverflowException: 'Arithmetic operation resulted in an overflow.

**Unchecked:** The Unchecked keyword works almost the same way as the default compiler works.

```
const int a = 2147483647;
const int b = 2147483647;
int c = (a + b);                    → It will give Compilation Error
Console.WriteLine(c);
```

If you want to bypass this behavior then you need to use the unchecked keyword.

```
const int a = 2147483647;
const int b = 2147483647;
int c = unchecked(a + b);
Console.WriteLine(c);
```

## Boxing and Unboxing

**Boxing:** Implicit **Conversion of a value type to a reference type** is called Boxing.

```
int num1 = 10;                  //value type
object obj = num1;              //Implicit Conversion from value type to reference type.
i = 20;
Console.WriteLine(obj);
Console.WriteLine(num1);
Output:
10
20
```

**Unboxing:** Explicit Conversion of a reference type to a value type called Unboxing.

```
int i = 10;                 //value type
object obj = i;
int j= (int)obj;            //Explicit Conversion
Console.WriteLine(obj);
Console.WriteLine(j);
Output:
10
10
```

## Convert.ToString and ToString

Both methods are used to Convert a value to a string. The difference is the **Convert.ToString()** method handles null whereas the **ToString()** doesn't handle null.

if you declare a **string variable** and if you **don't assign any value** to that variable, then by default that variable takes a **null** value. In such a case, if you use the **ToString()** method then your program will throw the **Null Reference Exception**. **On the other hand, if you use the Convert.ToString() method then your program will not throw an exception.**

```
string Name = null;
Name.ToString();                → it will throw an error  (null reference exception)
Console.ReadLine();
```

Reason : this is because the Name variable is null and we are invoking the ToString() method.

```
String Name = null;
Convert.ToString(Name);         → it should be executed without any error, so Convert.ToString handles the null.
Console.ReadLine();
```

## Stack and Heap Memory  / Value Type and Reference Type

When we declare a variable. It allocates some memory in RAM.

### 1: Value Type (Stack Memory/Struct)

Value types are types where the actual value is stored directly in memory.

- **Value stored in stack memory.**
- **Value type can not be null.**
- **Memory allocated at compile time.**
- **Example:** int, float, double, char, long, bool

```
struct Employee
{
   public int salary;
   public int Age;
}
class Program
{
   static void Main(string[] args)
   {
      Employee e= new Employee();
      e.salary = 10000;
      e.Age = 18;

      Employee e1 = e;
      Employee e2 = e;

      e.Age = 20;

      Console.WriteLine(e1.Age);
      Console.WriteLine(e2.Age);
   }
}
Output: 18
        18
```

### 2: Reference Type (Heap Memory /Class)

Reference types are types where the reference is stored of data.

- **Reference variable declared in stack memory, Object declared in Heap Memory.**
- **Reference type can be null.**
- **Memory allocated at run time.**
- **Example:** Class, Interfaces, Delegates, Arrays, strings

```
clas  Employee
{
   public int salary;
   public int Age;
}
class Program
{
   static void Main(string[] args)
   {
      Employee e= new Employee();
      e.salary = 10000;
```

```
        e.Age = 18;

        Employee e1 = e;
        Employee e2 = e;

        e.Age = 20;

        Console.WriteLine(e1.Age);
        Console.WriteLine(e2.Age);
    }
}
```

Output:
20
20

## Pass by Value and Pass by Reference

**1: Pass by Reference**

When you pass an argument to a method by value, you are passing a copy of the variable's value. This means that changes made to the parameter inside the method do not affect the original variable outside the method.

or

When a parameter is passed by value, a copy of the actual value is passed to the method. Therefore, any changes made to the parameter inside the method do not affect the original variable outside the method. This is the default behavior for value types in C# (like int, double, char, etc.).

```
using System;

class Program
{
    // Method that takes an argument by value
    static void ChangeValue(int number)
    {
        number = 100; // Change the value of 'number' inside the method
    }

    static void Main()
    {
        int originalValue = 10;
        Console.WriteLine("Before ChangeValue method: " + originalValue);

        ChangeValue(originalValue); // Passing by value

        Console.WriteLine("After ChangeValue method: " + originalValue); // The original value remains unchanged
    }
}
```

Before ChangeValue method: 10
After ChangeValue method: 10

**2: Pass by Reference:**

When you pass an argument to a method by reference, you are passing a reference to the original variable. This means that changes made to the parameter inside the method affect the original variable outside the method.

or

When a parameter is passed by reference, the method receives a reference to the original variable (not a copy of its value). This means that any changes made to the parameter inside the method will affect the original variable outside the method. You can pass a variable by reference in C# using the ref or out keywords.

```
using System;

class Program
{
    // Method that takes an argument by reference using 'ref' keyword
    static void ChangeValue(ref int number)
    {
        number = 100; // Modify the original variable directly
    }

    static void Main()
    {
        int originalValue = 10;
        Console.WriteLine("Before ChangeValue method: " + originalValue);

        ChangeValue(ref originalValue); // Passing by reference using 'ref'

        Console.WriteLine("After ChangeValue method: " + originalValue); // The original value changes
    }
}
```

Before ChangeValue method: 10
After ChangeValue method: 100

## Need of An Array

As we know a primitive type variable can hold only a single value at a time. **Example:** int num1 = 10;
If we want to store more than one value we need to use the concept of Array.

# Array

1. An array is a **collection of elements** (values) and it means it is used to store multiple elements of a single data type.
2. Each value referred to as an **element**.
3. Array works on indexing , first index number is 0 and last indexing number is n-1,
   (n =total number of elements)
4. Empty array is zero '0' by default.
5. Array are **reference type** variables, whose creation involves two steps:
   - Declaration= An array declaration specifies **data type** and **array name.**
   - Memory Location: Declaring an array does not allocate memory to the array. When it runs then allocate memory.

**Declaration of array:**
**Datatype[] arrayname- new data-type[size]**
**Example:** int [] num= new int [5];
5 is length of an array
Example: 23,45,66,74,343, only 5 elements will be here.

**Foreach Loop:** Foreach is an extension of for loop. It is used to perform actions on large data collections. Reads every element in the specified array.
Allow to execute a block of code in the array.

## Types of Arrays

1. Single-Dimensional Arrays (row and column are fixed)
2. Multi-Dimensional Arrays (row and column are fixed)
3. Jagged Array (row are fixed but column are not fixed)
4. Param Array

### Example1 : Single Integer Dimensional Arrays

```
int[] num= new int[4];        //4 is a length of array [45,67,39,94]
num[0] = 15;                  // new is a keyword which allocate memory in RAM
num[1] = 21;                  // num is array name (variable)
num[2] = 38;
num[3] = 49;
       or
int[] num = new int[4] { 15, 21, 38, 49 };
       or
int[] num = new int[] {15, 21, 38, 49};
       or
int[] num = {15, 21, 38, 49};

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

### For Loop

```
for (int i = 0; i < 4; i++)
for (int i = 0; i < num.length; i++)
for (int i = 0; i <=num.length-1; i++)
{
  Console.WriteLine(num[i]);
}
```

### Foreach

```
foreach(int item in num)
  {
      Console.WriteLine(item + " ");
  }

Console.WriteLine("length of array: " + num.Length);        //length of array
Console.WriteLine("max no. of array: " + num.Max());       //max no. in array
Console.WriteLine("min no. of array: " + num.Min());        //min no. in array
Console.WriteLine("rank of array: " + num.Rank);            //dimension of array
```

```
for (int i = 0; i < num.Length; i++)                                    //sum of elements
{
    sum = sum + num[i];
}
Console.WriteLine(sum);
```

Output:
15
21
38
49

4
49
15
1

123

---

Example 2: String Single Dimensional Array

```
string[] str1 = new string[5];            // 5 is a length of array
str1[0] = "Rohan";                           // str1 is  array name(variable)
str1[1] = "Shivam";
str1[2] = "Sudarshan";
str1[3] = "Vivek";
str1[4] = "Mayank";
      or
string[] str1 = new string[5] { "Rohan", "Shivam", "Sudarshan", "Vivek", "Mayank" };
      or
string[] str1 = new string[] { "Rohan", "Shivam", "Sudarshan", "Vivek", "Mayank" };

Console.WriteLine(str1[0]);
Console.WriteLine(str1[1]);
Console.WriteLine(str1[2]);
Console.WriteLine(str1[3]);
Console.WriteLine(str1[4]);
```

## For Loop

```
for (int i = 0; i < str1.Length; i++)
{
Console.WriteLine(str1[i]);
}
```

---

## Foreach

```
foreach(string item in str1)                                                    //we can used “var” instead of datatype.
{
 Console.WriteLine(item);
}

Console.WriteLine("Length of array is : {0}", str1.Length);                    //length
Array.Sort(str1);                                                              //sorting array
foreach (string i2 in str1)
{
Console.Write(i2+" ");
}
Console.WriteLine();

Array.Reverse(str1);                                                           //sorting reverse array
foreach (string i3 in str1)
{
Console.Write(i3+" ");
}
Console.ReadLine();


Output:
Rohan
Shivam
Sudarshan
Vivek
Mayank

Length of array is : 5
Rohan Shivam Sudarshan Vivek Mayank                    //print the array
Mayank Rohan Shivam Sudarshan Vivek                    //sorting the array
Vivek Sudarshan Shivam Rohan Mayank                    //reverse the array
--------------------------
```

### 2: Multi-Dimensional Arrays:

- Allows to store a combination of values of single data type in two or more dimension. And it represents rows and columns.
- It is also called an **array of arrays and 2D Arrays**.

### 1: Multi_Dimensional Rectangular Array

```
 int[,] num = new int[4, 4];
 for (int R = 0; R < 4; R++)
 {
 for (int C = 0; C < 4; C++)
 {
 Console.Write("[{0},{1}]", R, C);
 }
 Console.WriteLine();
 }
 Console.ReadLine();
```

```
Output:
[0,0][0,1][0,2][0,3]
[1,0][1,1][1,2][1,3]
[2,0][2,1][2,2][2,3]
[3,0][3,1][3,2][3,3]
```

Example 1:

```
int[,]  num = new int[4,4]
 {
       {34,65,87,34},
       {84,62,16,93},
       {35,74,24,73},
       {53,26,62,35}
 };
Console.Write(num[1, 1] +" ");
Console.Write(num[0, 3] + " ");
Console.Write(num[3, 1] + " ");
Console.Write(num[2, 3] + " ");
Console.ReadLine();
```

Output:

62 34 26 73

Example 2:

```
int[,] num = new int[4, 4]
{
{34, 65, 87, 34},
```

```
{84, 62, 16, 93},
{35, 74, 24, 73},
{53, 26, 62, 35}
};
for (int i=0; i<num.GetLength(0);i++)
{
   for (int j = 0; j < num.GetLength(1); j++)
     {
        Console.Write( num[i,j]+ " ");
     }
      Console.WriteLine();
}
Console.ReadLine();
```

Output:

34 65 87 34
84 62 16 93
35 74 24 73
53 26 62 35

## 2: Jagged Array

**(Nested For Loop, Foreach Loop, GetLength,Rank)**

```
int[][] num1 = new int[4][];
{
num1[0] = new[] { 34, 65, 87, 34, 45, 67, 34, };
num1[1] = new[] { 84, 62, 16, 93 ,56,23};
num1[2] = new[] { 35, 74, 24, 73, 76, 12, 34, 53, 76, 98 };
num1[3] = new[] { 53, 26, 62, 35 };
};
Console.WriteLine(num1[0][5]);
Console.WriteLine(num1[3][2]);
Console.WriteLine(num1[1][4]);
Console.WriteLine(num1[2][7]);
Console.ReadLine();
```

Output:

67
62
56
53

**Nested For Loop**

```
//jagged array
int[][] num1 = new int[4][];
{
num1[0] = new[] { 34, 65, 87, 34, 45, 67, 34, };
num1[1] = new[] { 84, 62, 16, 93, 56, 23 };
num1[2] = new[] { 35, 74, 24, 73, 76, 12, 34, 53, 76, 98 };
num1[3] = new[] { 53, 26, 62, 35 };
};
for (int i = 0; i < num1.Length; i++)
{
for (int j = 0; j < num1[i].Length; j++)
{
Console.Write(num1[i][j] + " ");
}
Console.WriteLine();
}
Console.ReadLine();
```

Output:

34 65 87 34 45 67 34
84 62 16 93 56 23
35 74 24 73 76 12 34 53 76 98
53 26 62 35

**Collections:** Collection is a set of data of different data type (set of heterogeneous elements) that can be modify at run-time.
Collection is a dynamic array.
Introduced In C# 1.0

There are two types of collection:

- Non-Generics Collections: Arraylist, Hashtable, stack, Queue (namespace: system.Collections)
- Generics Collections

**ArrayList:** ArrayList is a dynamic array. Used to store different type of value.
(Non-generic: can used to store all type of value),

```
using System.Collections;

ArrayList arraylist = new ArrayList();
arraylist.Add(10);
arraylist.Add("Hello");
arraylist.Add('d');
arraylist.Add(23.454);

for(int i=0;i<arraylist.Count;i++)
    or
foreach(var i in arraylist)
{
Console.WriteLine(arraylist[i]);
}
Console.ReadLine();
```

Output:
10
Hello
D

23.454

**Note:** Length works on same data type.
Count can on different data type

```
using System;
using System.Collections;
namespace collection1
{
    internal class Program
    {
        void ArrayLists()                                       //in class, declare method
        {
            ArrayList obj = new ArrayList();
            obj.Add("Hello");
            obj.Add("34");
            obj.Add("d");
            obj.Add("Hello");
            obj.Add("3223.44");

        Console.WriteLine("length: " + obj.Count);  //for count
        obj.Insert(2, "Hello World");                 //for insert element
        obj.Remove("Hello")                                 //for remove element
        obj.RemoveAt(1);                                        //remove the element on index
        obj.clear()                                                     //all element remove


        foreach( var item in obj)
            {
                Console.WriteLine(item);

            }

        }
        static void Main(string[] args)
        {

            Program program = new Program();
            program.ArrayLists();
            Console.ReadLine();
        }
    }
}
```

Hashtable:
Hashtable stores data in key/value format.
Array and arraylist has also key/value but index numbers.
Keys are user defined in hashtable.
Namespace: System.Collections
Use: where used to access elements using key.

```
Hashtable ht= new Hashtable();
ht.Add("ID", 1);
ht.Add("Name", "Rohan");
ht.Add("Salary", 12345.45);
ht.Add("Role", "Software Engineer");
ht.Add("Mobile", "9888393728");
ht.Add("Age", "26");
Console.WriteLine(ht["ID"]);
Console.WriteLine(ht["Name"]);
Console.WriteLine(ht["Salary"]);
Console.WriteLine(ht["Role"]);
Console.WriteLine(ht["Mobile"]);
Console.WriteLine(ht["Age"]);

Console.WriteLine();
foreach (object key  in ht.Keys)
{
Console.WriteLine(key + ": " + ht[key]);
}
```

Output:
Salary: 12345.45

Role: Software Engineer
Age: 26
ID: 1
Name: Rohan
Mobile: 9888393728