Chapter-1

Introduction to C# .NET.

What is C# ?

C# is pronounced "C-Sharp".

It is an object-oriented programming language created by Microsoft that runs on the .NET Framework.

C# has roots from the C family, and the language is close to other popular languages like C++ and Java.

The first version was released in year 2002. The latest version, C# 10.0, was released in September 2019.

C# is used for:

- Mobile applications
- Desktop applications
- Web applications
- Web services
- Web Sites
- Games
- VR
- Database applications
- And much, much more!

Why Use C#?

- It is one of the most popular programming language in the world.
- It is easy to learn and simple to use.
- It has a huge community support.
- C# is an object-oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- As C# is close to C, C++ and Java, it makes it easy for programmers to switch to C# or vice versa.

Development of C# .Net

History of C# language is interesting to know. Here we are going to discuss brief history of C# language.

C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.

Anders Hejlsberg is known as the founder of C# language.

It is based on **C++ and Java**, but it has many additional extensions used to perform component oriented programming approach.

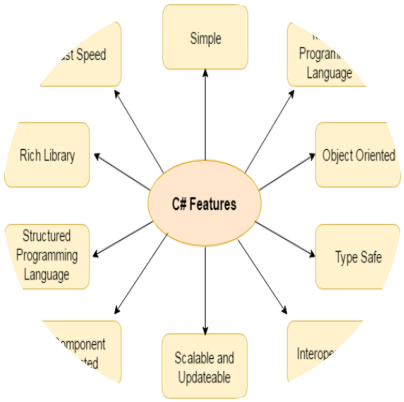
C# has evolved much since their first release in the year **2002**. It was introduced with **.NET Framework 1.0** and the current version of C# is **10.0 (Until This Note Is Prepare)**.

Version	Details	Year Released
1.0	C# v1.0 came with .NET framework 1.0,1.1 having CLR version 1.0 and Microsoft Visual Studio 2002.	2002
2.0	C# v2.0 came with .NET framework 2.0 having CLR version 2.0, and Microsoft Visual Studio 2005.	2005
3.0	C# v3.0 came with .NET framework 3.0,3.5 having CLR version 2.0, and Microsoft Visual Studio 2008.	2007
4.0	C# v4.0 came with .NET framework 4.0 having CLR version 4.0, and Microsoft Visual Studio 2010.	2010
5.0	C# v5.0 came with .NET framework 4.5 having CLR version 4.0, and Microsoft Visual Studio 2012, 2013.	2012
6.0	C# v6.0 came with .NET framework 4.6 having CLR version 4.0 and Microsoft Visual Studio 2013, 2015.	2015
7.0	C# v7.0 came with .NET framework 4.6, 4.6.1, 4.6.2 having CLR version 4.0, and Microsoft Visual Studio 2015 and 2017.	2017
8.0	C# v8.0 came with .NET framework 4.8 having CLR version 4.0, and Microsoft Visual Studio 2019.	2019
9.0	C# v9.0 came with .NET framework 5.0 having CLR version >4.*, and Microsoft Visual Studio 2020.	2020
10.0	C# v10.0 came with .NET framework6.0 having CLR version >4.*, and Microsoft Visual Studio 2020.	2022
11.0	Beta	2022

What is use of CLR?

> *The Common Language Runtime (CLR) is programming that manages the execution of programs written in any of several supported languages, allowing them to share common object-oriented classes written in any of the languages. It is a part of Microsoft's. NET Framework.*

Introduction to C# .net & it's features.



1. Simple

C# is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

2. Modern Programming Language

C# programming is based upon the current trend, and it is very powerful and simple for building scalable, interoperable and robust applications.

3. Object Oriented

C# is object-oriented programming language. OOPs, makes development and maintenance easier whereas in Procedure-oriented programming language it is not easy to manage if code grows as project size grow.

4. Type Safe

C# type safe code can only access the memory location that it has permission to execute. Therefore, it improves a security of the program.

5. Interoperability

Interoperability process enables the C# programs to do almost anything that a native C++ application can do.

6. Scalable and Updatable

C# is automatic scalable and updatable programming language. For updating our application we delete the old files and update them with new ones.

7. Component Oriented

C# is component oriented programming language. It is the predominant software development methodology used to develop more robust and highly scalable applications.

8. Structured Programming Language

C# is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

9. Rich Library

C# provides a lot of inbuilt functions that makes the development fast.

10. Fast Speed

The compilation and execution time of C# language is fast.

Basic structure of C# .Net

A C# program consists of the following parts:

- Namespace declaration
- A class
- Class methods
- Class attributes
- A Main method
- Statements and Expressions
- Comments

Program.cs

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Line 1:

`using System` means that we can use classes from the `System` namespace.

Line 2:

A blank line. C# ignores white space. However, multiple lines makes the code more readable.

Line 3:

`namespace` is used to organize your code, and it is a container for classes and other namespaces.

Line 4:

The curly braces `{}` marks the beginning and the end of a block of code.

Line 5:

`class` is a container for data and methods, which brings functionality to your program. Every line of code that runs in C# must be inside a class. In our example, we named the class Program.

Line 7:

Another thing that always appear in a C# program, is the Main method. Any code inside its curly brackets will be executed. You don't have to understand the keywords before and after `Main`. You will get to know them bit by bit while reading this tutorial.

Line 9:

The next line `/*...*/` is ignored by the compiler and it is put to add **comments** in the program.

Line 10:

`Console` is a class of the `System` namespace, which has a `WriteLine()` method that is used to output/print text. In our example it will output "Hello World!".

If you omit the `using System line`, you would have to write `System.Console.WriteLine()` to print/output text.

Note: *Every C# statement ends with a semicolon ;.*
Note: *C# is case-sensitive: "MyClass" and "myclass" has different meaning.*
Note: *Unlike Java, the name of the C# file does not have to match the class name, but they often do (for better organization). When saving the file, save it using a proper name and add ".cs" to the end of the filename. To run the example above on your computer, make sure that C# is properly installed: Go to the Get Started Chapter for how to install C#. The output should be:*

Variables:

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable.

The basic value types provided in C# can be categorized as:

Type	Example
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned
Nullable types	Nullable data types

C# also allows defining other value types of variable such as enum and reference types of variables such as class, which we will cover in subsequent chapters.

Defining Variables:

Syntax for variable definition in C# is:

```
<data_type> <variable_list>;
```

Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variable_list may consist of one or more identifier names separated by commas.

Some valid variable definitions are shown here:

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

Initializing Variables:

Variables are initialized (assigned a value) with an equal sign followed by a constant expression.

The general form of initialization is:

```
variable_name = value;
```

Variables can be initialized in their declaration. The initializer consists of an equal sign followed by a constant expression as −

```
<data_type> <variable_name> = value;
```

Some examples are

```
int d = 3, f = 5;    /* initializing d and f. */
byte z = 22;         /* initializes z. */
double pi = 3.14159; /* declares an approximation of pi. */
char x = 'x';        /* the variable x has the value 'x'. */
```

C# Identifiers:

Identifiers are the name given to entities such as variables, methods, classes, etc. They are tokens in a program which uniquely identify an element. For example,

```
int value;
```

Here, value is the name of variable. Hence it is an identifier. Reserved keywords can not be used as identifiers unless `@` is added as prefix example `@if` is valid Identifier but `if` is not valid identifier because it is a keyword, `int break;`

This statement will generate an error in compile time.

Rules for Naming an Identifier

- An identifier can not be a C# keyword.
- An identifier must begin with a letter, an underscore or `@` symbol. The remaining part of identifier can contain letters, digits and underscore symbol.
- Whitespaces are not allowed. Neither it can have symbols other than letter, digits and underscore.
- Identifiers are case-sensitive. So, `getName`, `GetName and getname` represents 3 different identifiers.

Here are some of the valid and invalid identifiers:

Identifier	Remarks
number	valid
calculateMarks	Valid
hello$	Invalid (Contains $)
name1	valid
@if	Valid (Keyword with prefix @)
if	Invalid (C# keyword)
my_name	Valid
My Name	Invalid (Contains whitespace)
_hello_hi	Valid
my@name	Invalid
my-name	Invalid

Example:

```
using System;

namespace HelloWorld
{
    class Hello
{
         static void Main(string[] args)
         {
           Console.WriteLine("Hello World!");
         }
    }
}
```

Keywords	Identifiers
using	System
namespace	HelloWorld (namespace)
class	Hello (Class)
static	Main (Method0
void	args
string	Console
	WriteLine