C# Tutorial:

01. What is C#:

C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.
By the help of C# programming language, we can develop different types of secured and robust applications:
Window applications, Web applications, Distributed applications , Web service applications, Database applications etc.
The most recent stable version of the language is C# 12.0, which was released in 2023 in .

---

02. C# Example: Hello World

In C# programming language, a simple "hello world" program can be written by multiple ways. Let's see the top 4 ways to create a simple C# example:

C# Simple Example

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

Description

class: is a keyword which is used to define class.
Program: is the class name. A class is a blueprint or template from which objects are created. It can have data members and methods. Here, it has only Main method.
static: is a keyword which means object is not required to access static members. So it saves memory.
void: is the return type of the method. It does't return any value. In such case, return statement is not required.
Main: is the method name. It is the entry point for any C# program. Whenever we run the C# program, Main() method is invoked first before any other method. It represents start up of the program.
string[] args: is used for command line arguments in C#. While running the C# program, we can pass values. These values are known as arguments which we can use in the program.
System.Console.WriteLine("Hello World!"): Here, System is the namespace. Console is the class defined in System namespace. The WriteLine() is the static method of Console class which is used to write the text on the console.

C# Example: Using System

If we write *using System* before the class, it means we don't need to specify System namespace for accessing any class of this namespace. Here, we are using Console class without specifying System.Console.

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

C# Example: Using public modifier

We can also specify *public* modifier before class and Main() method. Now, it can be accessed from outside the class also.

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

C# Example: Using namespace

We can create classes inside the namespace. It is used to group related classes. It is used to categorize classes so that it can be easy to maintain.

```
using System;
namespace ConsoleApplication1
{
public class Program
{
public static void Main(string[] args)
{
   Console.WriteLine("Hello World!");
} } }
```

---

03. C# Variable

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

The basic variable type available in C# can be categorized as:

Variable Type	Example
Decimal types	-> decimal
Boolean types	-> True or false value, as assigned
Integral types	-> int, char, byte, short, long
Floating point types	-> float and double
Nullable types	-> Nullable data types

Let's see the syntax to declare a variable:

```
type variable_list;
```

The example of declaring variable is given below:

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

Here, i, j, d, f, ch are variables and int, double, float, char are data types.
We can also provide values while declaring the variables as given below:

```
int i=2,j=4;  //declaring 2 variable of integer type
float f=40.2;
char ch='B';
```

Rules for defining variables:

A variable can have alphabets, digits and underscore.

A variable name can start with alphabet and underscore only. It can't start with digit.

No white space is allowed within variable name.

A variable name must not be any reserved word or keyword e.g. char, float etc.

Valid variable names:

```
int x;
int _x;
int k20;
```

Invalid variable names:

```
int 4;
int x y;
int double;
```

---

04. C# Data Types

A data type specifies the type of data that a variable can store such as integer, floating, character etc.

There are 3 types of data types in C# language.

Types	Data Types
Value Data Type	short, int, char, float, double etc
Reference Data Type	String, Class, Object and Interface
Pointer Data Type	Pointers

4.1. Value Data Type

The value data types are integer-based and floating-point based. C# language supports both signed and unsigned literals.

There are 2 types of value data type in C# language.

1) Predefined Data Types - such as Integer, Boolean, Float, etc.

2) User defined Data Types - such as Structure, Enumerations, etc.

The memory size of data types may change according to 32 or 64 bit operating system.

4.2. Reference Data Type

The reference data types do not contain the actual data stored in a variable, but they contain a reference to the variables.

If the data is changed by one of the variables, the other variable automatically reflects this change in value.

There are 2 types of reference data type in C# language.

1) Predefined Types - such as Objects, String.

2) User defined Types - such as Classes, Interface.

Example:

```
object  obj1 = new object();
object  obj2 = obj1; // Both obj1 and obj2 reference the same object in memory
obj1 = new object(); // Now obj1 references a new object, but obj2 still references the original object
```

```
string str1 = "Hello";
string str2 = str1; // Both str1 and str2 reference the same string object
str1 = "World"; // Now str1 references a new string object, but str2 still references "Hello"
```

In C#, int is a value type, not a reference type. This means that when you assign an int variable to another int variable, you are copying the value, not the reference. Changes to one variable do not affect the other. Here is an example:

```
int a = 5;
int b = a; // b gets a copy of the value of a, which is 5
a = 10; // Changing a does not affect b
Console.WriteLine(b); // Outputs 5
```

When a value type is converted to object type, it is called boxing and on the other hand, when an object type is converted to a value type, it is called unboxing.

```
object obj;
obj = 100; // this is boxing
```

Dynamic Type

You can store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time.

Syntax for declaring a dynamic type is –

dynamic <variable_name> = value;

For example,

```
dynamic d = 20;
```

Dynamic types are similar to object types except that type checking for object type variables takes place at compile time, whereas that for the dynamic type variables takes place at run time.

4.3. Pointer Data Type

The pointer in C# language is a variable, it is also known as locator or indicator that points to an address of a value.

Symbols used in pointer

Symbol	Name	Description
& (ampersand sign)	Address operator	Determine the address of a variable.
* (asterisk sign)	Indirection operator	Access the value of an address.

Example:

In C#, pointers can be used within an `unsafe` context. Pointers are variables that store the memory address of another variable. Here is an example demonstrating the use of pointers, the address operator (`&`), and the indirection operator (`*`):

```
using System;

class Program
{
    unsafe static void Main()
    {
        int number = 42;
        int* pointer = &number; // Pointer variable stores the address of number

        Console.WriteLine("Address of number: " + (long)pointer);
        Console.WriteLine("Value of number using pointer: " + *pointer);

        *pointer = 84; // Changing the value at the address
        Console.WriteLine("New value of number: " + number);
    }
}
```

Explanation

1. **Address Operator (`&`)**:
    - `&number` gets the address of the variable `number`.
2. **Indirection Operator (`*`)**:
    - `*pointer` accesses the value stored at the address held by `pointer`.

Output

```
Address of number: 140726536618884
Value of number using pointer: 42
```

```
New value of number: 84
```

---

5. C# operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.

There are following types of operators to perform different types of operations in C# language.

1) Binary Operator
    a. Arithmetic Operator (+,-,*,/,%)
    b. Relational Operator (<, >, <=, >=, !=, ==)
    c. Logical Operator (&&, ||, !)
    d. Bitwise Operator
    e. Assignment Operator (=, +=, -=, /=, *=, %=)
2) Unary Operator (++.--)
3) Ternary or Conditional Operator (?:)

5.1.Precedence of Operators in C#

The precedence of operator specifies that which operator will be evaluated first and next. The associativity specifies the operators direction to be evaluated, it may be left to right or right to left.

Let's understand the precedence by the example given below:

```
int data= 10+ 5*5
```

The "data" variable will contain 35 because * (multiplicative operator) is evaluated before + (additive operator).

5.2.Left-to-Right Associativity

Left-to-right associativity means that operations are performed from left to right. Most operators, including arithmetic and logical operators, have left-to-right associativity.

Example 1: Addition and Subtraction

Consider the expression: `5 - 2 + 3`

- **Step-by-Step Evaluation**:
    1. Evaluate `5 - 2` first (because of left-to-right associativity):
       `5 - 2 = 3`
    2. Then, evaluate `3 + 3`:
       `3 + 3 = 6`

So, the expression 5 - 2 + 3 evaluates to 6.

Right-to-Left Associativity

Right-to-left associativity means that operations are performed from right to left. Assignment operators and unary operators (like increment ++, decrement --) have right-to-left associativity.

Example 2: Assignment Operators

Consider the expression: x = y = z = 10

- **Step-by-Step Evaluation**:
    1. First, evaluate `z = 10` (assign `10` to `z`):
       `z = 10`
    2. Next, evaluate `y = z` (assign the value of `z` to `y`):
       `y = 10`
    3. Finally, evaluate `x = y` (assign the value of `y` to `x`):
       `x = 10`

---

6. Type Conversion

Type conversion is converting one type of data to another type. It is also known as Type Casting. In C#, type casting has two forms −

- **Implicit type conversion** − These conversions are performed by C# in a type-safe manner. For example, are conversions from smaller to larger integral types and conversions from derived classes to base classes.
- **Explicit type conversion** − These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

Explanation

1. **ToBoolean**: Converts a string to a boolean value.
2. **ToByte**: Converts a string to a byte.
3. **ToChar**: Converts an integer to a character.
4. **ToDateTime**: Converts a string to a `DateTime` object.
5. **ToDecimal**: Converts a double to a decimal.
6. **ToDouble**: Converts a string to a double.
7. **ToInt16**: Converts a string to a 16-bit integer.
8. **ToInt32**: Converts a string to a 32-bit integer.
9. **ToInt64**: Converts a string to a 64-bit integer.
10. **ToSByte**: Converts a string to a signed byte.
11. **ToSingle**: Converts a string to a single-precision floating point number.
12. **ToString**: Converts an integer to a string.
13. **ToType**: Converts a string to a specified type (in this case, double).
14. **ToUInt16**: Converts a string to an unsigned 16-bit integer.
15. **ToUInt32**: Converts a string to an unsigned 32-bit integer.
16. **ToUInt64**: Converts a string to an unsigned 64-bit integer.