Csharp for beginners:

C# Introduction:

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# 12**, was released in November 2023.

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 languages in the world
- It is easy to learn and simple to use
- It has 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

C# Syntax:

Simple program:

Program.cs

```
using System;


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

Result:

```
Hello World!
```

Example explained

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: `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.

C# Output

To output values or print text in C#, you can use the `WriteLine()` method:

Example

```
Console.WriteLine("Hello World!");
```

You can add as many `WriteLine()` methods as you want. Note that it will add a new line for each method:

Example

```
Console.WriteLine("Hello World!");

Console.WriteLine("I am Learning C#");

Console.WriteLine("It is awesome!");
```

You can also output numbers, and perform mathematical calculations:

Example

```
Console.WriteLine(3 + 3);
```

C# Comments

Comments can be used to explain C# code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Single-line Comments

Single-line comments start with two forward slashes (`//`).

Any text between `//` and the end of the line is ignored by C# (will not be executed).

This example uses a single-line comment before a line of code:

Example

```
// This is a comment

Console.WriteLine("Hello World!");
```

This example uses a single-line comment at the end of a line of code:

Example

```
Console.WriteLine("Hello World!");  // This is a comment
```

C# Multi-line Comments

Multi-line comments start with `/*` and ends with `*/`.

Any text between `/*` and `*/` will be ignored by C#.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/* The code below will print the words Hello World

to the screen, and it is amazing */
```

C# Variables

Variables are containers for storing data values.

In C#, there are different **types** of variables (defined with different keywords), for example:

- `int` - stores integers (whole numbers), without decimals, such as 123 or -123
- `double` - stores floating point numbers, with decimals, such as 19.99 or -19.99
- `char` - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- `string` - stores text, such as "Hello World". String values are surrounded by double quotes
- `bool` - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Syntax

```
type variableName = value;
```

Where *type* is a C# type (such as `int` or `string`), and *variableName* is the name of the variable (such as **x** or **name**). The **equal sign** is used to assign values to the variable.

To create a variable that should store text, look at the following example:

Example

Create a variable called **name** of type `string` and assign it the value "**John**":

```
string name = "John";

Console.WriteLine(name);
```

To create a variable that should store a number, look at the following example:

Example

Create a variable called **myNum** of type `int` and assign it the value **15**:

```
int myNum = 15;

Console.WriteLine(myNum);
```

You can also declare a variable without assigning the value, and assign the value later:

Example

```
int myNum;

myNum = 15;

Console.WriteLine(myNum);
```

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

Change the value of `myNum` to 20:

```
int myNum = 15;

myNum = 20; // myNum is now 20

Console.WriteLine(myNum);
```

Other Types

A demonstration of how to declare variables of other types:

Example

```
int myNum = 5;

double myDoubleNum = 5.99D;

char myLetter = 'D';

bool myBool = true;

string myText = "Hello";
```

C# Constants

Constants

If you don't want others (or yourself) to overwrite existing values, you can add the `const` keyword in front of the variable type.

This will declare the variable as "constant", which means unchangeable and read-only:

Example

```
const int myNum = 15;

myNum = 20; // error
```

C# Display Variables

Display Variables

The `WriteLine()` method is often used to display variable values to the console window.

To combine both text and a variable, use the `+` character:

Example

```
string name = "John";

Console.WriteLine("Hello " + name);
```

You can also use the `+` character to add a variable to another variable:

Example

```
string firstName = "John ";

string lastName = "Doe";

string fullName = firstName + lastName;

Console.WriteLine(fullName);
```

For numeric values, the `+` character works as a mathematical operator (notice that we use `int` (integer) variables here):

Example

```
int x = 5;

int y = 6;

Console.WriteLine(x + y); // Print the value of x + y
```

From the example above, you can expect:

- x stores the value 5
- y stores the value 6
- Then we use the `WriteLine()` method to display the value of x + y, which is **11**

C# Multiple Variables

Declare Many Variables

To declare more than one variable of the **same type**, use a comma-separated list:

Example

```
int x = 5, y = 6, z = 50;

Console.WriteLine(x + y + z);
```

You can also assign the **same value** to multiple variables in one line:

Example

```
int x, y, z;

x = y = z = 50;

Console.WriteLine(x + y + z);
```

C# Identifiers

C# Identifiers

All C# **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

Example

```
// Good

int minutesPerHour = 60;



// OK, but not so easy to understand what m actually is

int m = 60;
```

The general rules for naming variables are:

- Names can contain letters, digits and the underscore character (_)
- Names must begin with a letter or underscore
- Names should start with a lowercase letter, and cannot contain whitespace
- Names are case-sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like C# keywords, such as `int` or `double`) cannot be used as names.

C# Data Types

As explained in the variables chapter, a variable in C# must be a specified data type:

Example

```
int myNum = 5;               // Integer (whole number)

double myDoubleNum = 5.99D;  // Floating point number

char myLetter = 'D';         // Character

bool myBool = true;          // Boolean

string myText = "Hello";     // String
```

A data type specifies the size and type of variable values.

It is important to use the correct data type for the corresponding variable; to avoid errors, to save time and memory, but it will also make your code more maintainable and readable. The most common data types are:

Data Type	Size	Description
`int`	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,3
`float`	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digit
`double`	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
`bool`	1 bit	Stores true or false values
`char`	2 bytes	Stores a single character/letter, surrounded by single quotes
`string`	2 bytes per character	Stores a sequence of characters, surrounded by double quotes

---

Numbers

Number types are divided into two groups:

Integer types stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are `int` and `long`. Which type you should use, depends on the numeric value.

Floating point types represents numbers with a fractional part, containing one or more decimals. Valid types are `float` and `double`.

Integer Types

Int

The `int` data type can store whole numbers from -2147483648 to 2147483647. In general, and in our tutorial, the `int` data type is the preferred data type when we create variables with a numeric value.

Example

```
int myNum = 100000;

Console.WriteLine(myNum);
```

Long

The `long` data type can store whole numbers from -9223372036854775808 to 9223372036854775807. This is used when `int` is not large enough to store the value. Note that you should end the value with an "L":

Example

```
long myNum = 15000000000L;

Console.WriteLine(myNum);
```

Floating Point Types

You should use a floating point type whenever you need a number with a decimal, such as 9.99 or 3.14515.

The `float` and `double` data types can store fractional numbers. Note that you should end the value with an "F" for floats and "D" for doubles:

Float Example

```
float myNum = 5.75F;

Console.WriteLine(myNum);
```

Double Example

```
double myNum = 19.99D;

Console.WriteLine(myNum);
```

Use `float` or `double`?

Scientific Numbers

A floating point number can also be a scientific number with an "e" to indicate the power of 10:

Example

```
float f1 = 35e3F;

double d1 = 12E4D;

Console.WriteLine(f1);

Console.WriteLine(d1);
```

C# Type Casting

Type casting is when you assign a value of one data type to another type.

In C#, there are two types of casting:

- **Implicit Casting** (automatically) - converting a smaller type to a larger type size
  `char` -> `int` -> `long` -> `float` -> `double`
- **Explicit Casting** (manually) - converting a larger type to a smaller size type
  `double` -> `float` -> `long` -> `int` -> `char`

Implicit Casting

Implicit casting is done automatically when passing a smaller size type to a larger size type:

Example

```
int myInt = 9;

double myDouble = myInt;       // Automatic casting: int to double



Console.WriteLine(myInt);      // Outputs 9

Console.WriteLine(myDouble);   // Outputs 9
```

Explicit Casting

Explicit casting must be done manually by placing the type in parentheses in front of the value:

Example

```
double myDouble = 9.78;

int myInt = (int) myDouble;    // Manual casting: double to int



Console.WriteLine(myDouble);   // Outputs 9.78

Console.WriteLine(myInt);      // Outputs 9
```

Type Conversion Methods

It is also possible to convert data types explicitly by using built-in methods, such as `Convert.ToBoolean`, `Convert.ToDouble`, `Convert.ToString`, `Convert.ToInt32` (`int`) and `Convert.ToInt64` (`long`):

Example

```
int myInt = 10;

double myDouble = 5.25;

bool myBool = true;


Console.WriteLine(Convert.ToString(myInt));    // convert int to string

Console.WriteLine(Convert.ToDouble(myInt));    // convert int to double

Console.WriteLine(Convert.ToInt32(myDouble));  // convert double to int

Console.WriteLine(Convert.ToString(myBool));   // convert bool to string
```

C# User Input

Get User Input

In the following example, the user can input his or hers username, which is stored in the variable `userName`. Then we print the value of `userName`:

Example

```
// Type your username and press enter

Console.WriteLine("Enter username:");


// Create a string variable and get user input from the keyboard and store
it in the variable

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

```
// Print the value of the variable (userName), which will display the
input value

Console.WriteLine("Username is: " + userName);
```

User Input and Numbers

The `Console.ReadLine()` method returns a `string`. Therefore, you cannot get information from another data type, such as `int`. The following program will cause an error:

Example

```
Console.WriteLine("Enter your age:");

int age = Console.ReadLine();

Console.WriteLine("Your age is: " + age);
```

The error message will be something like this:

```
Cannot implicitly convert type 'string' to 'int'
```

Like the error message says, you cannot implicitly convert type 'string' to 'int'.

Luckily, for you, you just learned from the previous chapter (Type Casting), that you can convert any type explicitly, by using one of the `Convert.To` methods:

Example

```
Console.WriteLine("Enter your age:");

int age = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Your age is: " + age);
```

C# Operators

Operators

Operators are used to perform operations on variables and values.

In the example below, we use the `+` **operator** to add together two values:

Example

```
int x = 100 + 50;
```

Although the `+` operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

Example

```
int sum1 = 100 + 50;        // 150 (100 + 50)

int sum2 = sum1 + 250;      // 400 (150 + 250)

int sum3 = sum2 + sum2;     // 800 (400 + 400)
```

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations:

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y

-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

Assignment Operators

Assignment operators are used to assign values to variables.

In the example below, we use the **assignment** operator (=) to assign the value **10** to a variable called **x**:

Example

```
int x = 10;
```

The **addition assignment** operator (+=) adds a value to a variable:

Example

```
int x = 10;

x += 5;
```

A list of all assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3

>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either `True` or `False`. These values are known as *Boolean values*, and you will learn more about them in the Booleans and If..Else chapter.

In the following example, we use the **greater than operator** (>) to find out if 5 is greater than 3:

Example

```
int x = 5;

int y = 3;

Console.WriteLine(x > y); // returns True because 5 is greater than 3
```

A list of all comparison operators:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y

>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

As with comparison operators, you can also test for `True` or `False` values with **logical operators**.

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns True if both statements are true	x < 5 && x < 1
||	Logical or	Returns True if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns False if the result is true	!(x < 5 && x <