# C# Tutorials

C# is a simple & powerful object-oriented programming language developed by Microsoft. C# can be used to create various types of applications, such as web, windows, console applications or other types of applications using Visual studio.

# C# Version History

C# is a simple & powerful object-oriented programming language developed by Microsoft.

C# has evolved much since its first release in 2002. C# was introduced with .NET Framework 1.0 and the current version of C# is 6.0.

The following table lists important features introduced in each version of C#:

| Version | .NET Framework | Visual Studio | Important Features |
| --- | --- | --- | --- |
| C# 1.0 | .NET Framework 1.0/1.1 | Visual Studio .NET 2002 | * Basic features |
| C# 2.0 | .NET Framework 2.0 | Visual Studio 2005 | * Generics * Partial types * Anonymous methods * Iterators * Nullable types * Private setters (properties) * Method group conversions (delegates) * Covariance and Contra-variance * Static classes |
| C# 3.0 | .NET Framework 3.0\3.5 | Visual Studio 2008 | * Implicitly typed local variables * Object and collection initializers * Auto-Implemented properties * Anonymous types * Extension methods * Query expressions * Lambda expressions * Expression trees * Partial Methods |
| C# 4.0 | .NET Framework 4.0 | Visual Studio 2010 | * Dynamic binding (late binding) * Named and optional arguments * Generic co- and contravariance * Embedded interop types |
| C# 5.0 | .NET Framework 4.5 | Visual Studio 2012/2013 | * Async features * Caller information |
| C# 6.0 | .NET Framework 4.6 | Visual Studio 2013/2015 | * Expression Bodied Methods * Auto-property initializer * nameof Expression * Primary constructor * Await in catch block * Exception Filter * String Interpolation |
| C# 7.0 | .NET Core | Visual Studio 2017 | * out variables * Tuples * Discards * Pattern Matching * Local functions * Generalized async return types * throw Expressions |

# First C# Program

In the previous section, we have created a console project. Now, let's write simple C# code to understand important building blocks.

Every console application starts from the Main() method of Program class. The following example code displays "Hello World!!" on the console.

Example: Simple Console Project with C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CSharpTutorials

{

class Program

{

static void Main(string[] args)

{

string message = "Hello World!!";

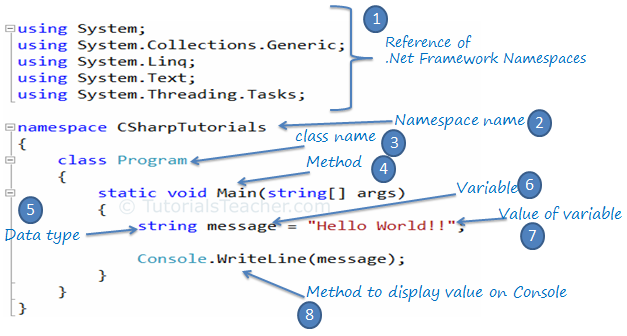
Console.WriteLine(message);

}

}

}

The following image illustrates the important parts of the above example.

[](http://www.tutorialsteacher.com/Content/images/csharp/csharp-code-structure.png)C# Code Structure

Let's understand the above C# structure.

1. Every .NET application takes the reference of the necessary .NET framework namespaces that it is planning to use with the "using" keyword e.g. using System.Text
2. Declare the namespace for the current class using the "namespace" keyword e.g. namespace CSharpTutorials.FirstProgram
3. We then declared a class using the "class" keyword: class Program
4. The Main() is a method of Program class which is the entry point of the console application.
5. String is a data type.
6. 'message' is a variable, that holds a value of a specified data type.
7. "Hello World!!" is the value of the message variable.
8. Console is a .NET framework class. WriteLine() is a method which you can use to display messages to the console.

C# Data Types

In the previous section, we have seen that a variable must be declared with the data type because C# is a strongly-typed language. For example,

string message = "Hello World!!";

Above, string is a data type, message is a variable, and "Hello World!!" is a string value assigned to a variable message.

Example: Data types

class Program

{

static void Main(string[] args)

{

string stringVar = "Hello World!!";

int intVar = 100;

float floatVar = 10.2f;

char charVar = 'A';

bool boolVar = true;

}

}

data type:

| Reserved Word | .NET Type | Type | Size (bits) | Range (values) |
| --- | --- | --- | --- | --- |
| byte | Byte | Unsigned integer | 8 | 0 to 255 |
| sbyte | SByte | Signed integer | 8 | -128 to 127 |
| short | Int16 | Signed integer | 16 | -32,768 to 32,767 |
| ushort | UInt16 | Unsigned integer | 16 | 0 to 65,535 |
| int | Int32 | Signed integer | 32 | -2,147,483,648 to 2,147,483,647 |
| uint | UInt32 | Unsigned integer | 32 | 0 to 4294967295 |
| long | Int64 | Signed integer | 64 | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| ulong | UInt64 | Unsigned integer | 64 | 0 to 18,446,744,073,709,551,615 |
| float | Single | Single-precision floating point type | 32 | -3.402823e38 to 3.402823e38 |
| double | Double | Double-precision floating point type | 64 | -1.79769313486232e308 to 1.79769313486232e308 |
| decimal | Decimal | Precise fractional or integral type that can represent decimal numbers with 29 significant digits | 128 | (+ or -)1.0 x 10e-28 to 7.9 x 10e28 |
| char | Char | A single Unicode character | 16 | Unicode symbols used in text |
| bool | Boolean | Logical Boolean type | 8 | True or False |
| object | Object | Base type of all other types |  |  |
| string | String | A sequence of characters |  |  |
| DateTime | DateTime | Represents date and time |  | 0:00:00am 1/1/01 to 11:59:59pm 12/31/9999 |

## Alias vs .Net Type

In the above table of data types, first column is for data type alias and second column is actual .Net type name. For example, int is an alias (or short name) for Int32. Int32 is a [structure](http://www.tutorialsteacher.com/csharp/csharp-struct) defined in System namespace. The same way, string represent String class.

| Alias | Type Name | .Net Type |
| --- | --- | --- |
| byte | System.Byte | struct |
| sbyte | System.SByte | struct |
| int | System.Int32 | struct |
| uint | System.UInt32 | struct |
| short | System.Int16 | struct |
| ushort | System.UInt16 | struct |
| long | System.Int64 | struct |
| ulong | System.UInt64 | struct |
| float | System.Single | struct |
| double | System.Double | struct |
| char | System.Char | struct |
| bool | System.Boolean | struct |
| object | System.Object | Class |
| string | System.String | Class |
| decimal | System.Decimal | struct |
| DateTime | System.DateTime | struct |

# C# Operators

Operator in C# is a special symbol that specifies which operations to perform on operands. For example, in mathematics the plus symbol (+) signifies the sum of the left and right numbers. In the same way, C# has many operators that have different meanings based on the data types of the operands. C# operators usually have one or two operands. Operators that have one operand are called Unary operators.

The following table list some of the operators available in C#.

| Operator category | Operators |
| --- | --- |
| Primary | x.y |
| Unary | +x |
| Multiplicative | x \* y |
| Additive | x + y |
| Shift | x << y |
| Relational and type testing | x < y |
| Equality | x == y |
| Logical AND | x & y |
| Logical XOR | x ^ y |
| Logical OR | x | y |
| Conditional AND | x && y |
| Conditional OR | x || y |
| Null-coalescing | x ?? y |
| Conditional | ?: |
| Assignment and lambda expression | x = y |

# if Statement

C# provides many decision making statements that help the flow of the C# program based on certain logical conditions. C# includes the following decision making statements.

1. if statement
2. if-else statement
3. switch statement
4. Ternary operator :?

Here, you will learn about the if statements.

Syntax:

if(boolean expression)

{

// execute this code block if expression evalutes to true

}

else if

static void Main(string[] args)

{

int i = 10, j = 20;

if (i > j)

{

Console.WriteLine("i is greater than j");

}

else if (i < j)

{

Console.WriteLine("i is less than j");

}

else

{

Console.WriteLine("i is equal to j");

}

}

# C# - switch

C# includes another decision making statement called switch. The switch statement executes the code block depending upon the resulted value of an expression.

Syntax:

switch(expression)

{

case <value1>

// code block

break;

case <value2>

// code block

break;

case <valueN>

// code block

break;

default

// code block

break;

}

# C# - for loop

The **for** keyword indicates a loop in C#. The for loop executes a block of statements repeatedly until the specified condition returns false.

Syntax:

for (variable initialization; condition; steps)

{

//execute this code block as long as condition is satisfied

}

As per the syntax above, the for loop contains three parts: initialization, conditional expression and steps, which are separated by a semicolon.

1. variable initialization: Declare & initialize a variable here which will be used in conditional expression and steps part.
2. condition: The condition is a boolean expression which will return either true or false.
3. steps: The steps defines the incremental or decremental part

Consider the following example of a simple for loop.

Example: for loop

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

{

Console.WriteLine("Value of i: {0}", i);

}

# C# - while loop:

C# includes the while loop to execute a block of code repeatedly.

Example: while loop in C#

int i = 0;

while (i < 10)

{

Console.WriteLine("Value of i: {0}", i);

i++;

}

# C# - do while

The do-while loop is the same as a 'while' loop except that the block of code will be executed at least once, because it first executes the block of code and then it checks the condition.

Example: do while loop

int i = 0;

do

{

Console.WriteLine("Value of i: {0}", i);

i++;

} while (i < 10);