C# For Loop – Syntax & Examples

C# For Loop

C# for loop is used to repeat a set of statements until a given condition is evaluated to False.

Syntax of C# For

```
for(initialization; boolean_expression; increment_decrement_update){
     /* statement(s) */
 }
```

where

- `for` is the keyword.
- `initialization` can have variable declarations which will be used inside for loop.
- `boolean_expression` can be a complex condition that evaluates to a boolean value: `True` or `False`.
- `increment_decrement_update` can have statements to increment, decrement or update the variables that control the iteration of loop execution.
- `statement(s)` are a set of statements that are meant to be executed in loop until the `boolean_expression` evaluates to false.

Example 1 – C# For Loop

Following is a basic example of C# for loop. We print numbers from 3 to 7.

Program.cs

```
using System;

namespace CSharpExamples {
    class Program {
        static void Main(string[] args) {
            for(int i=3;i<=7;i++) {
                Console.WriteLine(i);
            }
        }
    }
}
```

Output

```
PS D:\workspace\csharp\HelloWorld> dotnet run
3
4
5
6
7
```

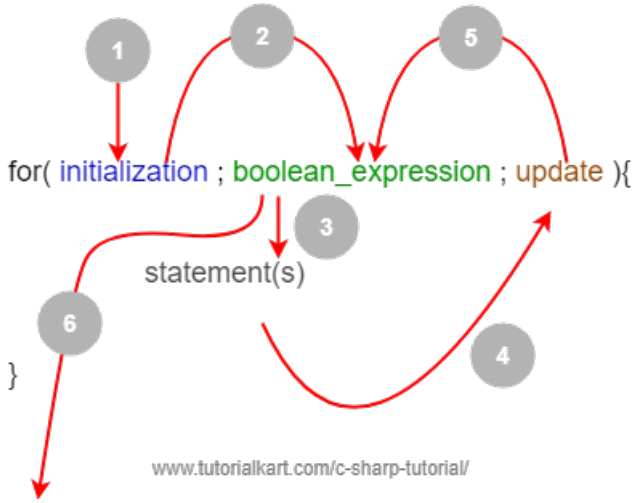
How C# for loop works?

Let us see how the for loop worked in the above example.

The intialization is `int i=3`. Meaning, we declared a variable `i` with an initial value of 3. We can access this variable `i` inside the for loop, which means at boolean_expression, increment_decrement_update and statement(s).

The boolean expression is `i<=7`. Therefore, we are going to come out of the for loop when this condition returns False.

The update expression is `i++`. So, during each iteration, `i` is incremented by `1`.



When the program control comes to the for loop, the initialization is executed **:Step 1** . Then it checks the boolean expression **:Step 2**. If it is true, then the control enters the for loop and executes the set of statements **:Step3**. Then the update happens: `i` is incremented **:Step4**. The boolean expression is evaluated **:Step 5**. If True, the statement(s) are executed **:Step3**. Then again the update happens, the boolean expression is evaluated and loop continues.

So, when does the execution come out of for loop. During this looping, when the evaluation of boolean expression returns False, that is when the program control comes out of the for loop **:Step 6**. And we are done executing the for loop.

Example 2 - C# For Loop with Decrement

Example 2 – C# For Loop with Decrement

In the following example, we will write a C# for loop with decrement as update. And print numbers from 8 to 4.

Program.cs

```
using System;

namespace CSharpExamples {
    class Program {
        static void Main(string[] args) {
            for(int i=8;i>=4;i--) {
                Console.WriteLine(i);
            }
        }
    }
}
```

Output

```
PS D:\workspace\csharp\HelloWorld> dotnet run
8
7
6
5
4
```

Example 3 – C# For Loop without Update

In the following example, we will write a C# for loop without any update to variables that control the boolean expression, in the for loop definition.

Program.cs

```
using System;

namespace CSharpExamples {
    class Program {
        static void Main(string[] args) {
            for(int i=8;i>=4;) {
                Console.WriteLine(i);
                i--;
            }
        }
    }
}
```

We do not have update section in the for loop definition. But inside the for loop, in statement(s) section, we have taken care of updating the variables.

If we do not take care of the update of the variables that control the output of boolean expression, it is very likely

If we do not take care of the update of the variables that control the output of boolean expression, it is very likely that our for loop ends up as an infinite loop.

Example 4 – C# For Loop without Intialization

In the following example, we will write a C# for loop with initialization section in for loop. We are using the variables that are outside the for loop. Also, in this example, we define two update statement in the update section of for loop. Yeah! You can have multiple initializations and multiple updates in your for loop definition.

Program.cs

```
using System;

namespace CSharpExamples {
    class Program {
        static void Main(string[] args) {
            int a=2;
            int b=3;
            for( ; a+b<=8; b=b+2, a--) {
                Console.WriteLine(a+b);
            }
        }
    }
}
```

Output

```
PS D:\workspace\csharp\HelloWorld> dotnet run
5
6
7
8
```

Example 5 – C# For Loop without Boolean Expression or Condition

Boolean expression in the for loop is also optional. But you have to take care of how to break the loop using break statement.

Program.cs

```
using System;

namespace CSharpExamples {
    class Program {
        static void Main(string[] args) {
            int b=3;
            for( ; ; b=b+2) {
                if(b>10){
                    break;
                }
```

```
                Console.WriteLine(b);
            }
        }
    }
}
```

Output

```
PS D:\workspace\csharp\HelloWorld> dotnet run
3
5
7
9
```

This is an early example for break statement. But we will in detail about that in our subsequent tutorials.

Nested For Loop

For loops can be nested. Meaning, we can place a for loop inside a for loop.

Example 6 – Nested For Loop

In the following example, we will write a for loop inside a for loop to print some decimal numbers.

Program.cs

```
using System;

namespace CSharpExamples {
    class Program {
        static void Main(string[] args) {
            for(int a=1; a<=4; a++) {
                for(int b=1; b<=2; b++) {
                    Console.WriteLine(a+"."+b);
                }
            }
        }
    }
}
```

Output

```
PS D:\workspace\csharp\HelloWorld> dotnet run
1.1
1.2
2.1
2.2
3.1
3.2
4.1
```

```
4.2
```

Conclusion

In this C# Tutorial, we learned the syntax of C# for loop, its operation, and different examples to understand its usage in and out.

- C# Encapsulation
- C# Polymorphism
- C# Method Overloading
- C# Interfaces

C# String Operations

- C# String Length
- C# Substring

C# File Operations

- C# Read Text File
- C# Write to File
- C# Delete File
- C# Copy File

C# Exception Handling

- C# try-catch
- C# finally
- C# throw
- C# Custom Exception
- C# SystemException
- C# DivideByZeroException
- C# NullReferenceException
- C# InvalidCastException
- C# IOException
- C# FieldAccessException

C# Collections

- C# List
- C# SortedList
- C# HashSet
- C# SortedSet
- C# Stack
- C# Queue

- C# LinkedList
- C# Dicionary
- C# SortedDictionary

C# Errors [Solved]

- C# Error: Class does not contain a constructor that takes arguments