C# – If, If..Else, If..Else If

There are three types of conditional statements using if and else keywords. A simple **if** statement, **if else** statement and then there is **if else if** statement.

C# If

If statement in C# is used to evaluate a set of statements conditionally based on an expression that evaluates to true or false.

Syntax

The syntax of C# If statement is:

```
if(boolean_expression) {
     /* statement(s) */
 }
```

The `boolean_expression` provided in the parenthesis after if keyword should evaluate to a boolean value, either `True` of `False`. If it evaluates to True, then the statement(s) inside the if block are executed. If the `boolean_expression` evaluates to False, then the statement(s) are not executed.

Example – C# If

In the following example, there are three if blocks demonstrating how an if statement is executed under different conditional evaluations.

Program.cs

```
using System;

namespace CSharpExamples
{
    class Program
    {
        static void Main(string[] args)
        {
            int a=1;
            int b=2;

            if(a == 1){
```

```
            if(a==1){
                Console.WriteLine("a is 1.");
            }

            if(a==b){
                Console.WriteLine("a is equal to b.");
            }

            if(a!=b){
                Console.WriteLine("a is not equal to b.");
            }
        }
    }
}
```

Output

```
PS D:\workspace\csharp\HelloWorld> dotnet run
a is 1.
a is not equal to b.
```

What happened in this C# If Statement Example?

There are three independent if blocks here.

In the first if block, the condition `a==1` evaluates to `True`. Therefore, the statements inside that if block are executed.

Coming to second if block example, the condition `a==b` evaluates to `False` and hence the statements inside this if block are not executed.

Finally in the third if block, the condition `a!=b` returns `True`. And the statements inside this if block are executed.

C# If..Else Statement

Program.cs

```
using System;

namespace CSharpExamples
{
    class Program
    {
        static void Main(string[] args)
        {
            int a=1;

            if(a==0){
                Console.WriteLine("a is 0.");
```

```
            } else if(a==1){
                Console.WriteLine("a is 1.");
            } else if(a==2){
                Console.WriteLine("a is 2.");
            }
        }
    }
}
```

Output

```
PS D:\workspace\csharp\HelloWorld> dotnet run
a is 1.
```

What happened in this C# If..Else Example?

In the above example, first if condition `a==0` is checked. It evaluates to `False`. Then, the control moves to the next if block `a==1` in the if-else-if ladder. The condition evaluates to `True`. Hence, the statements inside that if block are executed. The control then comes out of the whole if-else-if ladder without executing subsequent else-if blocks.

C# If..Else If Statement

Program.cs

```
using System;

namespace CSharpExamples
{
    class Program
    {
        static void Main(string[] args)
        {
            int a=1;
            int b=2;

            if(a==1){
                Console.WriteLine("a is 1.");
            }

            if(a==b){
                Console.WriteLine("a is equal to b.");
            }

            if(a!=b){
                Console.WriteLine("a is not equal to b.");
            }
        }
    }
}
```

Output

```
PS D:\workspace\csharp\HelloWorld> dotnet run
a is 1.
a is not equal to b.
```

Conclusion

In this C# Tutorial, we learned the syntax and usage of decision making statements: If, If-Else and If-Else-If.

- C# Class & Constructors
- 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