

C# Enum (Enumerator)

In `c#`, **enum** is a keyword that is useful to declare an enumeration. In `c#`, the enumeration is a type that consists of a set of named constants (/tutorial/csharp/csharp-const-constant-keyword) as a list.

By using an enumeration, we can group constants that are logically related to each other. For example, the days of the week can be grouped by using enumeration in `c#`.

C# Enum Syntax

Following is the syntax of defining an enumeration using `enum` keyword in `c#` programming language.

```
enum enum_name
{
    // enumeration list
}
```

If you observe the above syntax, we used `enum` keyword to define an enumeration based on our requirements.

Following is an example of defining an enumeration using `enum` keyword in `c#` programming language.

```
enum Week
{
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
}
```

If you observe the above example, we define an enumeration "**Week**" with a list of named constants (/tutorial/csharp/csharp-const-constant-keyword) called an enumeration list.

In `c#` by default, the first-named constant in the enumerator has a value of **0**, and the value of each successive item in the enumerator will be increased by **1**. For example, in the above enumeration, Sunday value is **0**, Monday is **1**, Tuesday is **2**, and so forth.

If we want to change the default values of an enumerator, then assigning a new value to the first item in the enumerator will automatically assign incremental values to the successive items in an enumerator.

Following is an example of overriding an enumerator's default values by assigning a new value to the first item in an enumerator.

```
enum Week
{
    Sunday = 10,
    Monday,
```

```
Tuesday,  
Wednesday,  
Thursday,  
Friday,  
Saturday  
}
```

In the above enumeration, the sequence of elements is forced to start from **10** instead of **0**, like Sunday value is **10**, Monday is **11**, Tuesday is **12**, and so forth.

C# Enum with Integer Types

In c#, an enumeration can contain only integral data types such as **byte**, **sbyte**, **short**, **ushort**, **int**, **uint**, **long** or **ulong**. It's impossible to use an enum with **string** or any other data types to define enum elements except numeric types.

The default type of enumeration element is **int** (integer). If you want to change the integral type of an enum to **byte**, you need to mention a **byte** type with a colon (:) after the identifier like as shown below.

```
enum Week: byte  
{  
    Sunday,  
    Monday,  
    Tuesday,  
    Wednesday,  
    Thursday,  
    Friday,  
    Saturday  
}
```

If you observe the above example, we are trying to change the default integral type of elements in an enumeration to **byte** type.

To get the enum elements' values in c#, an explicit cast is necessary to convert from an **enum** type to an integral type.

For example, the following are the statements to get an enum item value using a cast to convert from **enum** to **int**.

```
int a = (int)Week.Sunday; // It will return 0  
int b = (int)Week.Monday; // It will return 1
```

If you observe the above statements, we do cast conversion (**enum** to **int**) to get an enum item value.

C# Enum Example

Following is the example of declaring an enumeration using `enum` keyword in c# programming language.

```
using System;  
  
namespace Tutlane  
{  
    class Program  
    {  
        enum Week
```

```
{
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
}
static void Main(string[] args)
{
    int a = (int)Week.Sunday;
    int b = (int)Week.Monday;
    int c = (int)Week.Tuesday;
    Console.WriteLine(Week.Sunday);
    Console.WriteLine(Week.Monday);
    Console.WriteLine("Sunday: {0}", a);
    Console.WriteLine("Monday: {0}", b);
    Console.WriteLine("Tuesday: {0}", c);
    Console.WriteLine("\nPress Enter Key to Exit..");
    Console.ReadLine();
}
}
```

If you observe the above example, we defined an enumeration **Week** and got the enumeration items' values by explicitly converting from **enum** to **int** and assigned to integer variables.

When you execute the above c# program, you will get the result below.



If you observe the above result, the first enumerator (**Sunday**) has a value of **0**, and the value of each successive enumerator is increased by **1**.

C# Enum Methods

In c#, we have a class called **Enum** that contains the following helper methods to work with an enumeration (enum).

Method	Description
Format	It is useful to convert the value of the enum type to a specified string format.
GetName	It is useful to get the name of the specified enum item.
GetNames	It is useful to get all item names of the specified enum as an array.

Method	Description
GetValues	It is useful to get all item values of the specified enum as an array.
Parse	It is useful to convert the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.
GetUnderlyingType	It is useful to return the underlying type of the specified enumeration.

C# Iterate through Enum

In c#, we can iterate or loop through enum items using [for \(/tutorial/csharp/csharp-for-loop-with-examples\)](/tutorial/csharp/csharp-for-loop-with-examples) or [foreach \(/tutorial/csharp/csharp-foreach-loop-with-examples\)](/tutorial/csharp/csharp-foreach-loop-with-examples) loop to get the enumeration item names or values using enum helper methods.

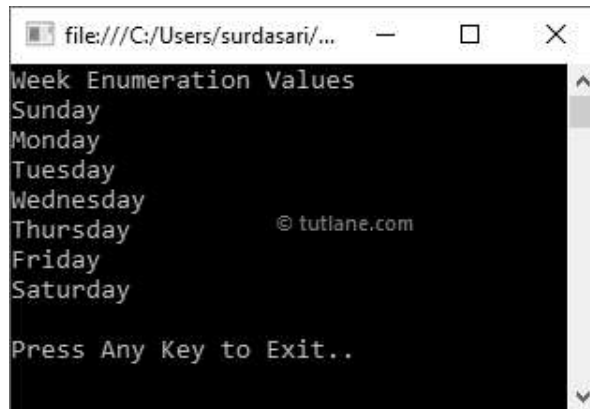
Following is the example of a loop through enum items to get all item names and values using enum helper methods based on our requirements.

```
using System;

namespace Tutlane
{
    class Program
    {
        enum Week
        {
            Sunday,
            Monday,
            Tuesday,
            Wednesday,
            Thursday,
            Friday,
            Saturday
        }
        static void Main(string[] args)
        {
            Console.WriteLine("Week Enumeration Values");
            foreach (string w in Enum.GetNames(typeof(Week)))
            {
                Console.WriteLine(w);
            }
            Console.WriteLine("\nPress Any Key to Exit..");
            Console.ReadLine();
        }
    }
}
```

If you observe the above example, we are looping through an enumeration using a [foreach \(/tutorial/csharp/csharp-foreach-loop-with-examples\)](/tutorial/csharp/csharp-foreach-loop-with-examples) loop and getting all item names using **GetNames()** helper method.

When you execute the above c# program, we will get the result below.



If you observe the above result, we are able to get all enumeration item names by looping through an enumeration using a foreach (/tutorial/csharp/csharp-foreach-loop-with-examples) loop based on our requirements.

In c#, it's better to define an enum within a namespace (/tutorial/csharp/csharp-namespaces-with-examples) so that all the classes (/tutorial/csharp/csharp-classes-and-objects-with-examples) in the namespace can access equally, and we can also nest an enum within a class (/tutorial/csharp/csharp-classes-and-objects-with-examples) or struct (/tutorial/csharp/csharp-structures-structs) based on our requirements.

CONTACT US

📍 **Address:** No.1-93, Pochamma Colony, Manikonda, Hyderabad, Telangana - 500089

✉ **Email:** support@tutlane.com (mailto:support@tutlane.com)