C# List

`List<T>` is a class that contains multiple objects of the same data type that can be accessed using an index. For example,

```
// list containing integer values
List<int> number = new List<int>() { 1, 2, 3 };
```

Here, `number` is a `List` containing integer values (**1**, **2** and **3**).

Create a List

To create `List<T>` in C#, we need to use the `System.Collections.Generic` namespace. Here is how we can create `List<T>`.For example,

```
using System;
using System.Collections.Generic;
class Program
{
    public static void Main()
    {
        // create a list named subjects that contain 2 elements
        List<string> subjects = new List<string>() { "English", "Math" };

    }
}
```

Access the List Elements

We can access `List` using index notation `[]`. For example,

```
using System;
using System.Collections.Generic;

class Program
{
    public static void Main()
    {
        // create a list
        List<string> languages = new List<string>() { "Python", "Java" };

        // access the first and second elements of languages list
        Console.WriteLine("The first element of the list is " + languages[0]);
        Console.WriteLine("The second element of the list is " + languages[1]);

    }
}
```

Output

```
The first element of the list is Python
The second element of the list is Java
```

Since the index of the list starts from **0**:

- `language[0]` - accesses the first element
- `language[5]` - accesses the fourth element

Iterate the List

In C#, we can also loop through each element of `List<T>` using a `for` loop. For example,

```
using System;
using System.Collections.Generic;

class Program
{
    public static void Main()
    {
        // create a list
        List<string> albums = new List<string>() { "Red", "Midnight", "Reputation" };

        // iterate through the albums list
        for (int i = 0; i < albums.Count; i++)

            Console.WriteLine(albums[i]);
    }
}
```

Output

```
Red
Midnight
Reputation
```

In the above example, we have looped through the `albums` list using a `for` loop.

Note: The `Count` property returns the total number of elements inside the list.

Basic Operations on List

The `List<T>` class provides various methods to perform different operations on `List`. We will look at some commonly used `List` operations in this tutorial:

- Add Elements
- Insert Elements

- Remove Elements

Let's discuss each operation in detail.

Add Elements to List

To add a single element to the `List`, we use the `Add()` method of the `List<T>` class. For example,

```
using System;
using System.Collections.Generic;

class Program
{
    public static void Main()
    {
        // create a list
        List<string> country = new List<string>() { "Russia" };

        //add "USA" to the country list
        country.Add("USA");


        // add "Japan" to the country list
        country.Add("Japan");


        // iterate through the country list
        for (int i = 0; i < country.Count; i++)
            Console.WriteLine(country[i]);
    }
}
```

Output

```
Russia
USA
Japan
```

In the above example, at first, we have created a `country` list that contains `"Russia"`.
Then we added `"USA"` and `"Japan"` to the list using the `Add()` method.

Insert Element in a List

To insert an element to a specified index in `List`, we use the `Insert()` method of the `List<T>` class. For example,

```
using System;
using System.Collections.Generic;

class Program
{
    public static void Main()
    {
        // create a list
        List<string> languages = new List<string>() { "Python", "Java", "C" };

        // insert "JavaScript" at index 2
        languages.Insert(2, "JavaScript");


        // display element at index 2
        Console.WriteLine(languages[2]);
    }
}
```

Output

```
JavaScript
```

In the above example,

- `languages.Insert(2, "JavaScript")` inserts `"JavaScript"` at the 2nd index position

Remove Elements from the List

We can delete one or more items from `List<T>` using **2** methods:

- `Remove()` - removes the first occurrence of an element from the given list
- `RemoveAt()` - removes the elements at the specified position in the list

Let's see examples using both methods.

Example: Remove() Method

```
using System;
using System.Collections.Generic;

class Program
{
    public static void Main()
    {
        var car = new List<string>() { "BMW", "Tesla", "Suzuki", "Tesla" };

        // remove the first occurence of "Tesla" from the list
        car.Remove("Tesla");
        // remove the first occurrence of "Suzuki"
        car.Remove("Suzuki");


        // print the updated list after removing
        for (int i = 0; i < car.Count; i++)
        {
            Console.WriteLine(car[i]);

        }
    }
}
```

Output

```
BMW
Tesla
```

Here,

- `car.Remove("Tesla")` - removes the first occurrence of `"Tesla"`
- `car.Remove("Suzuki")` - removes the first occurrence of `"Suzuki"`

We can see that,

- The original list: `{ "BMW", "Tesla", "Suzuki", "Tesla" }`
- The modified list: `{"BMW", "Tesla"}`

Example: RemoveAt() Method

```
using System;
using System.Collections.Generic;

class Program
{
    public static void Main()
    {
        var car = new List<string>() { "BMW", "Tesla", "Suzuki", "Tesla" };

        // remove the element present at the 2nd index position
        car.RemoveAt(2);



        // print the updated list after removing the element
        for (int i = 0; i < car.Count; i++)
        {
            Console.WriteLine(car[i]);

        }
    }
}
```

Output

```
BMW
Tesla
Tesla
```

In the above example, we have removed the element of `List<T>` using the `RemoveAt()` method.

Here, `car.RemoveAt(2)` removes `"Suzuki"` from the list.