

# C# List<T> Cheat Sheet for Coding Assessments

To use Lists, you must always include using System.Collections.Generic; at the top of your file.

## 1. Creating and Accessing

Unlike Arrays (which use .Length), Lists use .Count to find out how many items are inside.

- **List<int> nums = new List<int>();**: Creates an empty list.
- **List<int> nums = new List<int> { 1, 2, 3 };**: Creates a list with initial values.
- **nums.Count**: Gets the number of elements in the list. (*Remember: No parentheses!*)
- **nums[0]**: Accesses or modifies the first element (just like an array).

## 2. Adding Elements

- **list.Add(item)**: Adds an item to the very end of the list.
  - *Example*: nums.Add(4); -> { 1, 2, 3, 4 }
- **list.Insert(index, item)**: Squeezes an item into a specific index, pushing everything else to the right.
  - *Example*: nums.Insert(0, 99); -> { 99, 1, 2, 3 }
- **list.AddRange(anotherList)**: Adds multiple items (like an array or another list) to the end.

## 3. Removing Elements (The magic of Lists!)

- **list.Remove(item)**: Removes the *first occurrence* of that specific value. Returns true if it found it and removed it, false if the item wasn't there.
  - *Example*: nums.Remove(2); -> Removes the number 2.
- **list.RemoveAt(index)**: Removes whatever item is sitting at that exact index.
  - *Example*: nums.RemoveAt(0); -> Deletes the first item.
- **list.RemoveAll(condition)**: Removes ALL items that match a specific rule (using a Lambda expression).
  - *Example*: nums.RemoveAll(x => x < 0); -> Deletes all negative numbers.
- **list.Clear()**: Wipes out the entire list completely. Size becomes 0.

## 4. Searching and Checking

- **list.Contains(item)**: Returns true if the item exists in the list.
- **list.IndexOf(item)**: Returns the exact int index of the item. Returns -1 if it is not found.
- **list.Exists(condition)**: Returns true if any item matches a rule.
  - *Example*: nums.Exists(x => x > 100); -> True if there's a number over 100.

## 5. Sorting and Reversing

These methods change the list *in place* (they modify the actual list, they don't create a new one).

- **list.Sort():** Sorts the list in ascending order (A-Z or 0-9).
- **list.Reverse():** Flips the list completely backwards.

## 6. Conversions

You often have to jump back and forth between Arrays, Lists, and Strings.

- **list.ToArray():** Converts the List into a fixed-size Array.
- **array.ToList():** Converts an Array into a List (Requires using System.Linq;).
- **string.Join("", list):** Glues a List<string> or List<char> together into one giant String.



### Pro-Tip: Using List<char> for String Manipulation

If a problem asks you to heavily modify a string (delete specific characters, insert characters in the middle, etc.), converting it to a List<char> is the easiest way to do it:

```
string input = "User@2024!";
```

```
// 1. Convert string to List<char>
```

```
List<char> chars = input.ToList(); // Requires System.Linq
```

```
// 2. Easily remove special characters
```

```
chars.Remove('@');
```

```
chars.Remove('!');
```

```
// 3. Convert back to string
```

```
string cleanString = string.Join("", chars); // Result: "User2024"
```