C# collection types are designed to store, manage and manipulate similar data more efficiently. Data manipulation includes adding, removing, finding, and inserting data in the collection. Collection types implement the following common functionality:

* Adding and inserting items to a collection
* Removing items from a collection
* Finding, sorting, searching items
* Replacing items
* Copy and clone collections and items
* Capacity and Count properties to find the capacity of the collection and number of items in the collection

.NET supports two types of collections, generic collections and non-generic collections. Prior to .NET 2.0, it was just collections and when generics were added to .NET, generics collections were added as well.

Generic collections with work generic data type. Learn more about generics here:

The following table lists and matches these classes.

**Non-generic                          Generic**

 ArrayList     ------------->          List

 HashTable  ------------->          Dictionary

 SortedList   ------------->          SortedList

 Stack           ------------->          Stack

 Queue         ------------->          Queue

1. Non-Generic

In non-generic collections, each element can represent a value of a different type. The collection size is not fixed. Items from the collection can be added or removed at runtime. 

C# ArrayList

ArrayList class is a collection that can be used for any types or objects. 

1. Arraylist is a class that is similar to an array, but it can be used to store values of various types.
2. An Arraylist doesn't have a specific size.
3. Any number of elements can be stored.
4. using System.Collections;
5. **protected** **void** Button1\_Click(**object** sender, EventArgs e)
6. {
7. ArrayList al = **new** ArrayList();
8. **string** str = "kiran teja jallepalli";
9. **int** x = 7;
10. DateTime d = DateTime.Parse("8-oct-1985");
11. al.Add(str);
12. al.Add(x);
13. al.Add(d);
15. **foreach** (**object** o **in** al)
16. {
17. Response.Write(o);
18. Response.Write("<br>");
19. }
20. }

**Output**

kiran teja jallepalli  
7  
10/8/1985 12:00:00 AM

**Foreach Loop**

It executes for each and every item that exists in the arraylist object. Every time the loop rotates it reads one item from the arraylist and assignes it to the variable.

**Note:**Arraylist allocates memory for 4 items, whenever an object is created. When a fifth item is added, memory for another 4 items are added. it reduces the memory allocated for the object.

Capacity: is a property that returns the number of items for which memory is allocated.

Here is a detailed tutorial: [AarrayList in C#](https://www.c-sharpcorner.com/UploadFile/3d39b4/arraylist-in-C-Sharp/).

C# HashTable

HashTable is similar to arraylist but represents the items as a combination of a key and value.

1. using System.Collections;
3. **protected** **void** Button2\_Click(**object** sender, EventArgs e)
4. {
5. Hashtable ht = **new** Hashtable();
6. ht.Add("ora", "oracle");
7. ht.Add("vb", "vb.net");
8. ht.Add("cs", "cs.net");
9. ht.Add("asp", "asp.net");
11. **foreach** (DictionaryEntry d **in** ht)
12. {
13. Response.Write (d.Key + " " + d.Value);
15. Response.Write("<br>");
17. }
18. }

**Output**

vb  vb.net  
asp asp.net  
cs  cs.net  
ora oracle

**DictonaryEntry:**is a class whose object represents the data in a combination of key & value pairs.

C# SortedList

1. Is a class that has the combination of arraylist and hashtable.
2. Represents the data as a key and value pair.
3. Arranges all the items in sorted order.
4. using System.Collections;
5. **protected** **void** Button3\_Click(**object** sender, EventArgs e)
6. {
7. SortedList sl = **new** SortedList();
8. sl.Add("ora", "oracle");
9. sl.Add("vb", "vb.net");
10. sl.Add("cs", "cs.net");
11. sl.Add("asp", "asp.net");
13. **foreach** (DictionaryEntry d **in** sl)
14. {
15. Response.Write(d.Key + " " + d.Value);
16. Response.Write("<br>");
18. }
19. }

**Output**

asp  asp.net  
cs   cs.net  
ora  oracle  
vb   vb.net

Learn more here: [SortedList in C#.](https://www.c-sharpcorner.com/UploadFile/c25b6d/working-on-sortedlist-using-C-Sharp/)

C# Stack

1. **protected** **void** Button4\_Click(**object** sender, EventArgs e)
2. {
3. Stack stk = **new** Stack();
4. stk.Push("cs.net");
5. stk.Push("vb.net");
6. stk.Push("asp.net");
7. stk.Push("sqlserver");
9. **foreach** (**object** o **in** stk)
10. {
11. Response.Write(o + "<br>");
12. }
13. }

**Output**

sqlserver  
asp.net  
vb.net  
cs.net

Here is a detailed tutorial in [Stack in C#.](https://www.c-sharpcorner.com/UploadFile/c25b6d/working-on-stack-using-C-Sharp/)

C# Queue

1. using System.Collections;
2. **protected** **void** Button5\_Click(**object** sender, EventArgs e)
3. {
4. Queue q = **new** Queue();
5. q.Enqueue("cs.net");
6. q.Enqueue("vb.net");
7. q.Enqueue("asp.net");
8. q.Enqueue("sqlserver");
10. **foreach** (**object** o **in** q)
11. {
12. Response.Write(o + "<br>");
13. }
14. }

**Output**

cs.net  
vb.net  
asp.net  
sqlserver

Here is a detailed tutorial: [Queue in C#](https://www.c-sharpcorner.com/UploadFile/c25b6d/CSharp-Queue/)

2. Generic Collections

Generic Collections work on the specific type that is specified in the program whereas non-generic collections work on the object type.

1. Specific type
2. Array Size is not fixed
3. Elements can be added / removed at runtime.

C# List

1. **using** System.Collections.Generic;
3. **protected** **void** Button1\_Click(**object** sender, EventArgs e)
4. {
5. List<**int**> lst = **new** List<**int**>();
6. lst.Add(100);
7. lst.Add(200);
8. lst.Add(300);
9. lst.Add(400);
10. **foreach** (**int** i **in** lst)
11. {
12. Response.Write(i+"<br>");
13. }
14. }

C# Dictonary

1. **using** System.Collections.Generic;
3. **protected** **void** Button1\_Click(**object** sender, EventArgs e)
4. {
5. Dictionary<**int**, **string**> dct = **new** Dictionary<**int**, **string**>();
6. dct.Add(1, "cs.net");    +
7. dct.Add(2, "vb.net");
8. dct.Add(3, "vb.net");
9. dct.Add(4, "vb.net");
10. **foreach** (KeyValuePair<**int**, **string**> kvp **in** dct)
11. {
12. Response.Write(kvp.Key + " " + kvp.Value);
13. Response.Write("<br>");
14. }
15. }

C# SortedList

1. **using** System.Collections.Generic;
3. **protected** **void** Button3\_Click(**object** sender, EventArgs e)
4. {
5. SortedList<**string**, **string**> sl = **new** SortedList<**string**, **string**>();
6. sl.Add("ora", "oracle");
7. sl.Add("vb", "vb.net");
8. sl.Add("cs", "cs.net");
9. sl.Add("asp", "asp.net");
11. **foreach** (KeyValuePair<**string**, **string**> kvp **in** sl)
12. {
13. Response.Write(kvp.Key + " " + kvp.Value);
14. Response.Write("<br>");
15. }
16. }

C# Stack

1. **using** System.Collections.Generic;
3. **protected** **void** Button4\_Click(**object** sender, EventArgs e)
4. {
5. Stack<**string**> stk = **new** Stack<**string**>();
6. stk.Push("cs.net");
7. stk.Push("vb.net");
8. stk.Push("asp.net");
9. stk.Push("sqlserver");
11. **foreach** (**string** s **in** stk)
12. {
13. Response.Write(s + "<br>");
14. }
15. }

C# Queue

1. using System.Collections.Generic;
2. **protected** **void** Button1\_Click(**object** sender, EventArgs e)
3. {
4. Queue<**string**> q = **new** Queue<**string**>();
6. q.Enqueue("cs.net");
7. q.Enqueue("vb.net");
8. q.Enqueue("asp.net");
9. q.Enqueue("sqlserver");
11. **foreach** (**string** s **in** q)
12. {
13. Response.Write(s + "<br>");
14. }
15. }