

## 1. ARRAYS IN C#

### Definition:

An **array** is a collection of elements of the same data type stored in contiguous memory locations. It allows easy access to elements using an index.

### Syntax:

```
int[] numbers = new int[5];      // Declaration
```

```
int[] values = {1, 2, 3, 4, 5};    // Initialization
```

### Key Properties and Methods:

Property/Method	Description	Example
Length	Returns total number of elements	arr.Length
Sort()	Sorts elements in ascending order	Array.Sort(arr)
Reverse()	Reverses order of elements	Array.Reverse(arr)
IndexOf()	Returns index of specific value	Array.IndexOf(arr, value)

### Types of Arrays in C#

Type	Declaration	Description
<b>Single Dimensional</b>	int[] arr = new int[5];	One-dimensional (like list)
<b>Multi-Dimensional</b>	int[,] matrix = new int[2,3];	2D or 3D matrix-type arrays
<b>Jagged Array</b>	int[][] jagged = new int[3][];	Array of arrays (each sub-array can have different lengths)

### Example:

```
int[] marks = { 90, 85, 80 };
```

```
foreach (int mark in marks)
```

```
{
```

```
    Console.WriteLine(mark);
```

```
}
```

## 2. STRINGS IN C#

### Definition:

A **string** is a sequence of characters enclosed within double quotes.

In C#, strings are **immutable**, meaning their value cannot be changed after creation.

### Characteristics:

- Stored as objects of the System.String class.
- Supports operators like +, ==, and !=.
- Strings can be manipulated using various built-in methods.

### Common String Methods:

Method	Description	Example	Output
Length	Returns string length	"Hello".Length	5
ToUpper()	Converts to uppercase	"hello".ToUpper()	HELLO
ToLower()	Converts to lowercase	"HELLO".ToLower()	hello
Substring(start, len)	Extract substring	"Dhruv".Substring(0,3)	Dhr
Replace(old, new)	Replace characters	"C#".Replace("#","Sharp")	CSharp
Split(' ')	Splits string into array	"Hello World".Split(' ')	["Hello", "World"]
Trim()	Removes whitespace	" hello ".Trim()	"hello"
Contains()	Checks substring	"Hello".Contains("He")	True

### Example:

```
string company = "Dhruv Compusoft";
Console.WriteLine(company.ToUpper());
Console.WriteLine(company.Replace("Compusoft", "Technology"));
```

### String Interpolation

String interpolation makes it easy to embed variables directly inside strings.

```
string name = "Udaya";
Console.WriteLine($"Welcome {name} to .NET training!");
```

## **StringBuilder (Mutable String)**

For frequent string modifications, use StringBuilder (from System.Text).

```
using System.Text;
```

```
StringBuilder sb = new StringBuilder("Hello");
sb.Append(" World");
sb.Replace("World", "Dhruv");
Console.WriteLine(sb.ToString());
```

**Reason:** StringBuilder is mutable → better performance for loops or concatenations.

## **3. COLLECTIONS IN C#**

### **Definition:**

Collections are **dynamic data structures** that store, retrieve, and manipulate groups of related objects efficiently.

They are part of the **System.Collections** and **System.Collections.Generic** namespaces.

### **Types of Collections:**

Type	Description	Namespace	Example
<b>List&lt;T&gt;</b>	Dynamic array of specific type	System.Collections.Generic	List<int>
<b>Dictionary&lt; TKey, TValue &gt;</b>	Stores key-value pairs	System.Collections.Generic	Dictionary<string,int>
<b>Queue&lt;T&gt;</b>	FIFO (First In, First Out)	System.Collections.Generic	Queue<string>
<b>Stack&lt;T&gt;</b>	LIFO (Last In, First Out)	System.Collections.Generic	Stack<int>
<b>HashSet&lt;T&gt;</b>	Stores unique elements only	System.Collections.Generic	HashSet<int>

### **Example: List**

```
List<string> names = new List<string>() {"Udaya", "Shetty"};  
names.Add("Dhruv");  
foreach (string n in names)  
{  
    Console.WriteLine(n);  
}
```

### **Example: Dictionary**

```
Dictionary<string, int> ages = new Dictionary<string, int>();  
ages["Udaya"] = 24;  
ages["Suresh"] = 26;  
  
foreach (var pair in ages)  
{  
    Console.WriteLine($" {pair.Key} - {pair.Value}");  
}
```

### **Example: Queue & Stack**

```
Queue<int> queue = new Queue<int>();  
queue.Enqueue(1);  
queue.Enqueue(2);  
Console.WriteLine("Dequeued: " + queue.Dequeue());
```

```
Stack<int> stack = new Stack<int>();  
stack.Push(10);  
stack.Push(20);  
Console.WriteLine("Popped: " + stack.Pop());
```

### **Example: HashSet**

```
HashSet<int> set = new HashSet<int>() {1, 2, 2, 3};  
foreach (var item in set)  
{  
    Console.WriteLine(item);  
}
```

Output:

```
1  
2  
3
```

(duplicates are ignored)

## **4. EXCEPTION HANDLING IN C#**

### **Definition:**

An **exception** is a runtime error that disrupts program execution.

C# provides structured exception handling using try, catch, finally, and throw.

### **Syntax:**

```
try  
{  
    // Code that may throw exception  
}  
  
catch (Exception ex)  
{  
    Console.WriteLine(ex.Message);  
}  
  
finally  
{  
    Console.WriteLine("Execution finished.");  
}
```

## **Key Terms:**

<b>Keyword</b>	<b>Description</b>
try	Contains code that may cause an error
catch	Handles the exception
finally	Executes regardless of exception
throw	Manually raises an exception

## **Common Exception Types:**

<b>Exception</b>	<b>Description</b>
DivideByZeroException	Division by zero
NullReferenceException	Accessing a null object
IndexOutOfRangeException	Invalid array index
FormatException	Invalid input format
FileNotFoundException	File not found
InvalidOperationException	Invalid operation for state

## **Example:**

```
try {
    int x = 10, y = 0;
    int z = x / y;
}
catch (DivideByZeroException)
{
    Console.WriteLine("Error: Division by zero is not allowed.");
}
finally
{
    Console.WriteLine("Program execution complete.");
}
```

## Custom Exception Example

You can create your own exception by extending the Exception class.

```
class InvalidAgeException : Exception {  
    public InvalidAgeException(string message) : base(message) {}  
}  
  
class Demo {  
    static void Main()  
    {  
        try  
        {  
            int age = 15;  
            if (age < 18)  
                throw new InvalidAgeException("Age must be 18 or above.");  
        }  
        catch (InvalidAgeException ex)  
        {  
            Console.WriteLine("Custom Exception: " + ex.Message);  
        }  
    }  
}
```

## 5. THE VAR, DYNAMIC, AND OBJECT KEYWORDS

These are **data typing mechanisms** that define how variable types are decided.

Keyword	Type Decided	Compile Time Check	Example
<b>var</b>	At compile time	Yes	var x = 10;
<b>dynamic</b>	At runtime	No	dynamic a = 10; a = "Text";
<b>object</b>	Base type for all data	Needs casting	object obj = 10;

**Example:**

```
var a = 10;      // int  
dynamic b = "Hello"; // runtime  
object c = 25;    // must cast later: (int)c
```

## 6. FOREACH LOOP

Used to iterate over arrays, lists, or collections.

```
int[] arr = {10, 20, 30};  
  
foreach (int num in arr)  
{  
    Console.WriteLine(num);  
}
```

### Summary Table

Concept	Description	Example
<b>Array</b>	Fixed-size same-type data collection	int[] nums = new int[5];
<b>String</b>	Immutable text data	"Dhruv".ToUpper()
<b>StringBuilder</b>	Mutable string class	sb.Append("Text")
<b>List</b>	Dynamic array	List<int>
<b>Dictionary</b>	Key-value pairs	Dictionary<string,int>
<b>Stack</b>	LIFO collection	Push(), Pop()
<b>Queue</b>	FIFO collection	Enqueue(), Dequeue()
<b>Exception Handling</b>	Handling runtime errors	try-catch-finally
<b>Custom Exception</b>	User-defined errors	class MyException : Exception
<b>var/dynamic/object</b>	Variable typing system	Compile-time / Runtime

## Snapshots:

### 1. Single-Dimensional Array

The screenshot shows the Visual Studio Code interface with a dark theme. The left sidebar includes the Explorer, Open Editors, and Solution Explorer panes. The main editor pane displays a C# file named Program.cs containing code to sort an array of integers. The terminal at the bottom shows the output of running the program with the command `dotnet run`, displaying the marks list and sorted marks.

```
1 using System;
2
3 class ArrayDemo
4 {
5     static void Main()
6     {
7         int[] marks = { 85, 90, 75, 80, 95 };
8
9         Console.WriteLine("Marks List:");
10        foreach (int mark in marks)
11            Console.Write(mark + " ");
12
13        Console.WriteLine("\nSorted Marks:");
14        Array.Sort(marks);
15        foreach (int mark in marks)
16            Console.Write(mark + " ");
17    }
18 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty.NET\_Training\Week1\_CSharp\_SQL\Day-4\Day4Programs> dotnet run

>> Marks List:  
85 90 75 80 95  
Sorted Marks:  
75 80 85 90 95

PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty.NET\_Training\Week1\_CSharp\_SQL\Day-4\Day4Programs>

OUTLINE TIMELINE SOLUTION EXPLORER

0 0 Projects: Day4Programs Debug Any CPU Solution file opened: Day4Programs.sln Ask GPT

Ln 19, Col 1 Spaces: 4 UTF-8 with BOM CRLF ⌂ C# Go Live ⌂

### 2. Multi-Dimensional Array

The screenshot shows the Visual Studio Code interface with a dark theme. The left sidebar includes the Explorer, Open Editors, and Solution Explorer panes. The main editor pane displays a C# file named Program.cs containing code to print the elements of a 2D integer array. The terminal at the bottom shows the output of running the program with the command `dotnet run`, displaying the matrix elements.

```
20 using System;
21
22 class MatrixDemo
23 {
24     static void Main()
25     {
26         int[,] matrix = { { 1, 2, 3 }, { 4, 5, 6 } };
27
28         Console.WriteLine("Matrix Elements:");
29         for (int i = 0; i < 2; i++)
30         {
31             for (int j = 0; j < 3; j++)
32                 Console.Write(matrix[i, j] + " ");
33             Console.WriteLine();
34         }
35     }
36 }
37
38
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty.NET\_Training\Week1\_CSharp\_SQL\Day-4\Day4Programs> dotnet run

>> Matrix Elements:  
1 2 3  
4 5 6

PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty.NET\_Training\Week1\_CSharp\_SQL\Day-4\Day4Programs>

OUTLINE TIMELINE SOLUTION EXPLORER

0 0 Projects: Day4Programs Debug Any CPU Solution file opened: Day4Programs.sln Ask GPT

Ln 38, Col 1 Spaces: 4 UTF-8 with BOM CRLF ⌂ C# Go Live ⌂

### 3. Jagged Array

The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left lists files like 'Program.cs', 'Day4Programs.csproj', and 'Day4Programs.sln'. The main editor window displays the following C# code:

```
41 class JaggedArrayDemo
42 {
43     static void Main()
44     {
45         int[][] jagged = new int[3][];
46         jagged[0] = new int[] { 1, 2 };
47         jagged[1] = new int[] { 3, 4, 5 };
48         jagged[2] = new int[] { 6 };
49
50         for (int i = 0; i < jagged.Length; i++)
51         {
52             Console.WriteLine("Row " + i + ":");
53             foreach (int val in jagged[i])
54                 Console.WriteLine(val + " ");
55             Console.WriteLine();
56         }
57     }
58 }
```

The terminal at the bottom shows the output of running the program with 'dotnet run':

```
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-4\Day4Programs> dotnet run
>>>
Row 0: 1 2
Row 1: 3 4 5
Row 2: 6
```

### 4. String Operations

The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left lists files like 'Program.cs', 'Day4Programs.csproj', and 'Day4Programs.sln'. The main editor window displays the following C# code:

```
59
60 using System;
61
62 class StringDemo
63 {
64     static void Main()
65     {
66         string s = " Dhruv Compusoft Pvt Ltd ";
67
68         Console.WriteLine("Original: " + s + "'");
69         Console.WriteLine("Trimmed: " + s.Trim() + "'");
70         Console.WriteLine("Upper: " + s.ToUpper());
71         Console.WriteLine("Replace: " + s.Replace("Compusoft", "Technologies"));
72         Console.WriteLine("Substring(2,5): " + s.Substring(2,5));
73         Console.WriteLine($"Length: {s.Length}");
74     }
75 }
76
```

The terminal at the bottom shows the output of running the program with 'dotnet run':

```
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-4\Day4Programs> dotnet run
>>>
Original: ' Dhruv Compusoft Pvt Ltd '
Trimmed: 'Dhruv Compusoft Pvt Ltd'
Upper: DHRUV COMPUSOFT PVT LTD
Replace: Dhruv Technologies Pvt Ltd
Substring(2,5): Dhruv
Length: 27
```

## 5. StringBuilder Example

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project named "Day4Programs" with files "Program.cs" and "Program.csproj".
- Code Editor:** Displays the following C# code:

```
74 // }
75 //
76
77 using System;
78 using System.Text;
79
80 class StringBuilderDemo
81 {
82     static void Main()
83     {
84         StringBuilder sb = new StringBuilder("Welcome");
85         sb.Append(" to Dhruv");
86         sb.Append(" .NET Training");
87         sb.Replace("Dhruv", "Dhruv Compusoft");

88         Console.WriteLine(sb);
89     }
90 }
91 }
```

- Terminal:** Shows the command `dotnet run` being run, outputting "Welcome to Dhruv Compusoft .NET Training".
- Status Bar:** Shows the file is open at line 92, column 1, with 4 spaces, in UTF-8 with BOM, CRLF.

## 6. List Collection

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project named "Day4Programs" with files "Program.cs" and "Program.csproj".
- Code Editor:** Displays the following C# code:

```
95 using System.Collections.Generic;
96
97 class ListDemo
98 {
99     static void Main()
100    {
101        List<string> students = new List<string>() {"Udaya", "Kiran", "Arjun"};
102        students.Add("Shetty");

103        Console.WriteLine("Students:");
104        foreach (string s in students)
105            Console.WriteLine(s);

106        students.Remove("Kiran");
107        Console.WriteLine("\nAfter Removal:");
108        students.ForEach(Console.WriteLine);
109
110    }
111 }
112
113 }
```

- Terminal:** Shows the command `dotnet run` being run, outputting the names of the students and then removing Kiran from the list.
- Status Bar:** Shows the file is open at line 112, column 2, with 4 spaces, in UTF-8 with BOM, CRLF.

## 7. Dictionary Collection

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like Program.cs, Day4Programs.csproj, and Day4Programs.sln.
- Code Editor:** Displays the following C# code:

```
114  using System;
115  using System.Collections.Generic;
116
117  0 references
118  class DictionaryDemo
119  {
120      0 references
121      static void Main()
122      {
123          Dictionary<string, int> ages = new Dictionary<string, int>();
124          ages["Udaya"] = 24;
125          ages["Arun"] = 25;
126          ages["Kiran"] = 23;
127
128          foreach (var kv in ages)
129          {
130              Console.WriteLine($"{kv.Key} - {kv.Value}");
131          }
132      }
133  }
```
- Terminal:** Shows the command `dotnet run` being run, with the output:

```
>>>
Udaya - 24
Arun - 25
Kiran - 23
```
- Right Panel:** Features an "Ask about your code" AI integration and a list of multiple PowerShell windows.
- Bottom Status Bar:** Shows the file path as PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty.NET\_Training\Week1\_CSharp\_SQL\Day-4\Day4Programs, the line number as Line 131, and other standard status bar information.

## 8. Stack & Queue

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like Program.cs, Day4Programs.csproj, and Day4Programs.sln.
- Code Editor:** Displays the following C# code:

```
132
133  using System;
134  using System.Collections.Generic;
135
136  0 references
137  class StackQueueDemo
138  {
139      0 references
140      static void Main()
141      {
142          Stack<int> stack = new Stack<int>();
143          stack.Push(10);
144          stack.Push(20);
145          stack.Push(30);
146          Console.WriteLine("Stack Pop: " + stack.Pop());
147
148          Queue<string> queue = new Queue<string>();
149          queue.Enqueue("A");
150          queue.Enqueue("B");
151          queue.Enqueue("C");
152          Console.WriteLine("Queue Dequeue: " + queue.Dequeue());
153      }
154  }
```
- Terminal:** Shows the command `dotnet run` being run, with the output:

```
>>>
Stack Pop: 30
Queue Dequeue: A
```
- Right Panel:** Features an "Ask about your code" AI integration and a list of multiple PowerShell windows.
- Bottom Status Bar:** Shows the file path as PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty.NET\_Training\Week1\_CSharp\_SQL\Day-4\Day4Programs, the line number as Line 153, and other standard status bar information.

## 9. HashSet (Unique Items)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like Program.cs, Day4Programs.csproj, and Day4Programs.sln.
- Code Editor:** Displays the following C# code in Program.cs:

```
150     queue.Enqueue(1);
151     queue.Enqueue(2);
152     queue.Enqueue(3);
153     queue.Enqueue(4);
154
155     using System;
156     using System.Collections.Generic;
157
158     class HashSetDemo
159     {
160         static void Main()
161         {
162             HashSet<int> set = new HashSet<int>() {1, 2, 2, 3, 4};
163             Console.WriteLine("Unique Elements:");
164             foreach (int n in set)
165                 Console.WriteLine(n);
166         }
167     }
168
```

- Terminal:** Shows the output of running the program with the command `dotnet run`. The output is:

```
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-4\Day4Programs> dotnet run
>>>
Unique Elements:
1
2
3
4
```

- Status Bar:** Shows the current file is Program.cs, and other details like CPU, Solution file opened, and Ask GPT.

## 10. Exception Handling (try–catch–finally)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like Program.cs, Day4Programs.csproj, and Day4Programs.sln.
- Code Editor:** Displays the following C# code in Program.cs:

```
169     using System;
170
171     class ExceptionDemo
172     {
173         static void Main()
174         {
175             try
176             {
177                 Console.Write("Enter number1: ");
178                 int a = Convert.ToInt32(Console.ReadLine());
179                 Console.Write("Enter number2: ");
180                 int b = Convert.ToInt32(Console.ReadLine());
181                 int result = a / b;
182                 Console.WriteLine($"Result: {result}");
183             }
184             catch (DivideByZeroException)
185             {
186                 Console.WriteLine("Division by zero not allowed!");
187             }
188             catch (FormatException)
189             {
190                 Console.WriteLine("Please enter numeric values only!");
191             }
192             finally
193             {
194                 Console.WriteLine("Execution complete.");
195             }
196         }
197     }
198
```

- Terminal:** Shows the output of running the program with the command `dotnet run`. The output is:

```
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-4\Day4Programs> dotnet run
>>>
Enter number1: 10
Enter number2: 2
Result: 5
Execution complete.
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-4\Day4Programs> dotnet run
>>>
Enter number1: 10
Enter number2: 0
Division by zero not allowed!
Execution complete.
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-4\Day4Programs>
```

- Status Bar:** Shows the current file is Program.cs, and other details like CPU, Solution file opened, and Ask GPT.

## 11. Custom Exception

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like Program.cs, Day4Programs.csproj, and Day4Programs.sln.
- Code Editor:** Displays the content of Program.cs, which defines a custom exception class `InvalidAgeException` and a main method that handles it.
- Terminal:** Shows the command `dotnet run` being executed, followed by the output: "Enter age: 14" and "Custom Exception : Age must be 18 or above!"
- Solution Explorer:** Shows the project configuration.

```
201     3 references
202     class InvalidAgeException : Exception
203     {
204         1 reference
205         public InvalidAgeException(string message) : base(message) { }
206
207     0 references
208     class CustomExceptionDemo
209     {
210         0 references
211         static void Main()
212         {
213             try
214             {
215                 Console.WriteLine("Enter age: ");
216                 int age = Convert.ToInt32(Console.ReadLine());
217                 if (age < 18)
218                     throw new InvalidAgeException("Age must be 18 or above!");
219                 Console.WriteLine("You are eligible.");
220             }
221             catch (InvalidAgeException ex)
222             {
223                 Console.WriteLine("Custom Exception + " + ex.Message);
224             }
225         }
226     }
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
```

## 12. var / dynamic / object Demo

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like Program.cs, Day4Programs.csproj, and Day4Programs.sln.
- Code Editor:** Displays the content of Program.cs, which demonstrates the use of var, dynamic, and object keywords.
- Terminal:** Shows the command `dotnet run` being executed, followed by the output: "var: 10", "dynamic: Hello", "object: 25", and "dynamic new value: 123".
- Solution Explorer:** Shows the project configuration.

```
224     // }
225
226     using System;
227
228     0 references
229     class VariableDemo
230     {
231         0 references
232         static void Main()
233         {
234             var x = 10;           // compile-time type
235             dynamic y = "Hello"; // runtime type
236             object z = 25;       // base type
237
238             Console.WriteLine($"var: {x} ({x.GetType()})");
239             Console.WriteLine($"dynamic: {y} ({y.GetType()})");
240             Console.WriteLine($"object: {z} ({z.GetType()})");
241
242             y = 123; // valid for dynamic
243             Console.WriteLine($"dynamic new value: {y}");
244         }
245
```