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** |
| **Single Dimensional** | int[] arr = new int[5]; | One-dimensional (like list) |
| **Multi-Dimensional** | int[,] matrix = new int[2,3]; | 2D or 3D matrix-type arrays |
| **Jagged Array** | 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);

}

1. **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.

1. **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** |
| **List<T>** | Dynamic array of specific type | System.Collections.Generic | List<int> |
| **Dictionary<TKey,TValue>** | Stores key-value pairs | System.Collections.Generic | Dictionary<string,int> |
| **Queue<T>** | FIFO (First In, First Out) | System.Collections.Generic | Queue<string> |
| **Stack<T>** | LIFO (Last In, First Out) | System.Collections.Generic | Stack<int> |
| **HashSet<T>** | 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)

1. **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:**

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| 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:**

|  |  |
| --- | --- |
| **Exception** | **Description** |
| 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);

}

}

}

1. **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** |
| **var** | At compile time | Yes | var x = 10; |
| **dynamic** | At runtime | No | dynamic a = 10; a = "Text"; |
| **object** | 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

1. **FOREACH LOOP**

Used to iterate over arrays, lists, or collections.

int[] arr = {10, 20, 30};

foreach (int num in arr)

{

Console.WriteLine(num);

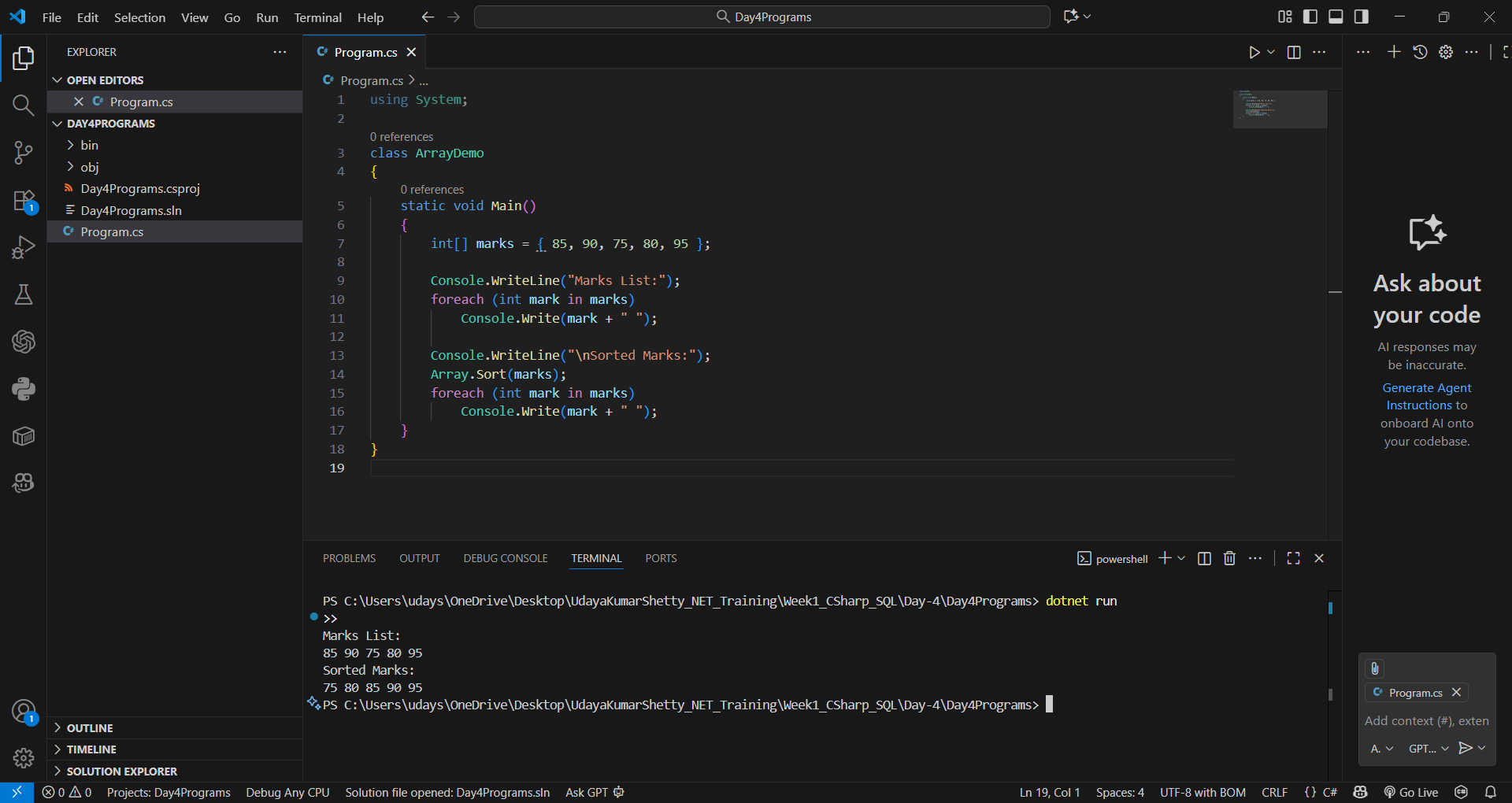
}

**Summary Table**

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

**Snapshots:**

1. **Single-Dimensional Array**

****

1. **Multi-Dimensional Array**

A screenshot of a computer

AI-generated content may be incorrect.

1. **Jagged Array**

A screenshot of a computer program

AI-generated content may be incorrect.

1. **String Operations**

A screenshot of a computer

AI-generated content may be incorrect.

1. **StringBuilder Example**

A screenshot of a computer

AI-generated content may be incorrect.

1. **List Collection**

A screenshot of a computer

AI-generated content may be incorrect.

1. **Dictionary Collection**

A screenshot of a computer

AI-generated content may be incorrect.

1. **Stack & Queue**

A screenshot of a computer program

AI-generated content may be incorrect.

1. **HashSet (Unique Items)**

A screenshot of a computer program

AI-generated content may be incorrect.

1. **Exception Handling (try–catch–finally)**

A screenshot of a computer

AI-generated content may be incorrect.

1. **Custom Exception**

A screenshot of a computer program

AI-generated content may be incorrect.

1. **var / dynamic / object Demo**

A screenshot of a computer program

AI-generated content may be incorrect.