C# Array Functions Examples (Single, Multi-Dimensional, Jagged)

# 1. Array.Length

Explanation:

Returns the number of elements in the array.

Code:

// Single-dimensional  
int[] temperatures = { 23, 19, 31, 27, 30, 25 };  
Console.WriteLine(temperatures.Length); // Output: 6  
  
// Multi-dimensional  
int[,] matrix = { {1, 2}, {3, 4} };  
Console.WriteLine(matrix.Length); // Output: 4  
  
// Jagged array  
int[][] jagged = { new int[] {1, 2}, new int[] {3, 4, 5} };  
Console.WriteLine(jagged.Length); // Output: 2 (outer array)

# 2. Array.Rank

Explanation:

Returns the number of dimensions in the array.

Code:

// Single-dimensional  
int[] numbers = { 10, 20, 30 };  
Console.WriteLine(numbers.Rank); // Output: 1  
  
// Multi-dimensional  
int[,] grid = new int[3, 4];  
Console.WriteLine(grid.Rank); // Output: 2  
  
// Jagged array  
int[][] jagged = new int[3][];  
Console.WriteLine(jagged.Rank); // Output: 1 (outer array is 1D)

# 3. Array.IndexOf

Explanation:

Finds the index of a value in a 1D array. Doesn't work with 2D arrays.

Code:

// Single-dimensional  
string[] fruits = { "Apple", "Banana", "Cherry" };  
int index = Array.IndexOf(fruits, "Banana");  
Console.WriteLine(index); // Output: 1  
  
// Jagged array (match by reference)  
int[] row1 = { 1, 2, 3 };  
int[] row2 = { 4, 5, 6 };  
int[][] jagged = { row1, row2 };  
int idx = Array.IndexOf(jagged, row2);  
Console.WriteLine(idx); // Output: 1  
  
// 2D array - Invalid  
int[,] matrix = { { 1, 2 }, { 3, 4 } };  
// Array.IndexOf(matrix, 3); // ❌ Compile-time error

# 4. Array.LastIndexOf

Explanation:

Finds the last index of a repeated value.

Code:

int[] nums = { 1, 2, 3, 2, 4 };  
int lastIndex = Array.LastIndexOf(nums, 2);  
Console.WriteLine(lastIndex); // Output: 3

# 5. Array.Find

Explanation:

Finds the first element that matches a condition.

Code:

int[] numbers = { 10, 15, 30, 25 };  
int result = Array.Find(numbers, n => n > 20);  
Console.WriteLine(result); // Output: 30

# 6. Array.FindIndex

Explanation:

Finds the index of the first element that matches a condition.

Code:

int[] numbers = { 5, 10, 15, 20 };  
int idx = Array.FindIndex(numbers, n => n > 10);  
Console.WriteLine(idx); // Output: 2

# 7. Array.Exists

Explanation:

Checks if any element matches a condition.

Code:

int[] ages = { 18, 21, 30 };  
bool exists = Array.Exists(ages, age => age > 25);  
Console.WriteLine(exists); // Output: True

# 8. Array.BinarySearch

Explanation:

Searches for a value in a sorted array.

Code:

int[] sortedArray = { 10, 20, 30, 40 };  
int position = Array.BinarySearch(sortedArray, 30);  
Console.WriteLine(position); // Output: 2

# 9. Array.Sort

Explanation:

Sorts the array.

Code:

int[] values = { 3, 1, 4, 2 };  
Array.Sort(values);  
Console.WriteLine(string.Join(", ", values)); // Output: 1, 2, 3, 4

# 10. Array.Reverse

Explanation:

Reverses the elements in the array.

Code:

int[] sequence = { 1, 2, 3 };  
Array.Reverse(sequence);  
Console.WriteLine(string.Join(", ", sequence)); // Output: 3, 2, 1

# 11. Array.Clear

Explanation:

Clears part of the array.

Code:

int[] nums = { 5, 10, 15 };  
Array.Clear(nums, 1, 2);  
Console.WriteLine(string.Join(", ", nums)); // Output: 5, 0, 0

# 12. Array.Copy

Explanation:

Copies elements from one array to another.

Code:

int[] source = { 1, 2, 3 };  
int[] target = new int[3];  
Array.Copy(source, target, 3);  
Console.WriteLine(string.Join(", ", target)); // Output: 1, 2, 3

# 13. Array.Resize

Explanation:

Changes the size of an array.

Code:

int[] arr = { 1, 2 };  
Array.Resize(ref arr, 4);  
Console.WriteLine(string.Join(", ", arr)); // Output: 1, 2, 0, 0

# 14. LINQ - Where

Explanation:

Filters elements based on a condition.

Code:

int[] numbers = { 1, 2, 3, 4 };  
var evens = numbers.Where(n => n % 2 == 0);  
Console.WriteLine(string.Join(", ", evens)); // Output: 2, 4

# 15. LINQ - Select

Explanation:

Transforms elements of the array.

Code:

int[] nums = { 1, 2, 3 };  
var squared = nums.Select(n => n \* n);  
Console.WriteLine(string.Join(", ", squared)); // Output: 1, 4, 9

# 16. LINQ - Sum, Max, Min, Average

Explanation:

Performs aggregate operations.

Code:

int[] values = { 2, 4, 6 };  
Console.WriteLine(values.Sum()); // Output: 12  
Console.WriteLine(values.Max()); // Output: 6  
Console.WriteLine(values.Min()); // Output: 2  
Console.WriteLine(values.Average()); // Output: 4

# 17. LINQ - All, Any

Explanation:

Checks conditions across array elements.

Code:

int[] scores = { 70, 80, 90 };  
Console.WriteLine(scores.All(s => s >= 60)); // Output: True  
Console.WriteLine(scores.Any(s => s == 100)); // Output: False

# 18. LINQ - Distinct

Explanation:

Removes duplicate elements.

Code:

int[] nums = { 1, 2, 2, 3 };  
var unique = nums.Distinct();  
Console.WriteLine(string.Join(", ", unique)); // Output: 1, 2, 3

# 19. LINQ - OrderBy / OrderByDescending

Explanation:

Sorts elements using LINQ.

Code:

int[] numbers = { 5, 3, 8 };  
var sorted = numbers.OrderBy(n => n);  
Console.WriteLine(string.Join(", ", sorted)); // Output: 3, 5, 8

# 20. LINQ - ToArray

Explanation:

Converts LINQ result to array.

Code:

int[] values = { 10, 20, 30 };  
var result = values.Where(v => v > 15).ToArray();  
Console.WriteLine(string.Join(", ", result)); // Output: 20, 30