**for (JavaScript)**

const arr = [1, 2, 3];

for (let i = 0; i < 5; i++) {

console.log(arr[i]);

}

------------------------------------------------------------------------------

const obj = { a: 1, b: 2, c: 3 };

for (let prop in obj) {

console.log(prop); // Logs 'a', 'b', 'c'

console.log(obj[prop]); // Logs 1, 2, 3

}

------------------------------------------------------------------------------

const arr = [1, 2, 3];

for (let value of arr) {

console.log(value); // Logs 1, 2, 3

}

------------------------------------------------------------------------------

const array = [1, 2, 3, 4, 5];

array.forEach(function(element) {

console.log(element);

});

**for (C#)**

int[] numbers = { 1, 2, 3, 4, 5 };

for (int i = 0; i < numbers.Length; i++)

{

Console.WriteLine(numbers[i]);

}

------------------------------------------------------------------------------

int[] numbers = { 1, 2, 3, 4, 5 };

foreach (int number in numbers)

{

Console.WriteLine(number);

}

**String**

using System;

public class Program

{

public static void Main(String[] args)

{

string original = " Hello, World! ";

// Length

Console.WriteLine("Length: " + original.Length);

// Substring

string substring = original.Substring(1, 5);

Console.WriteLine("Substring: " + substring);

// IndexOf

int indexOfW = original.IndexOf('W');

Console.WriteLine("IndexOf 'W': " + indexOfW);

// LastIndexOf

int lastIndexOfL = original.LastIndexOf('l');

Console.WriteLine("LastIndexOf 'l': " + lastIndexOfL);

// Contains

bool containsHello = original.Contains("Hello");

Console.WriteLine("Contains 'Hello': " + containsHello);

// Replace

string replaced = original.Replace("World", "C#");

Console.WriteLine("Replace 'World' with 'C#': " + replaced);

// ToUpper

string upper = original.ToUpper();

Console.WriteLine("ToUpper: " + upper);

// ToLower

string lower = original.ToLower();

Console.WriteLine("ToLower: " + lower);

// Trim

string trimmed = original.Trim();

Console.WriteLine("Trim: " + trimmed);

// Split

string[] split = original.Split(' ');

Console.WriteLine("Split by space: " + string.Join(", ", split));

// Join

string joined = string.Join("-", split);

Console.WriteLine("Join with '-': " + joined);

// StartsWith

bool startsWithHello = original.StartsWith(" Hello");

Console.WriteLine("StartsWith ' Hello': " + startsWithHello);

// EndsWith

bool endsWithExclamation = original.EndsWith("! ");

Console.WriteLine("EndsWith '! ': " + endsWithExclamation);

// Insert

string inserted = original.Insert(7, "beautiful ");

Console.WriteLine("Insert 'beautiful ': " + inserted);

// Remove

string removed = original.Remove(7, 5);

Console.WriteLine("Remove 5 characters at index 7: " + removed);

// PadLeft

string padLeft = original.PadLeft(20, '\*');

Console.WriteLine("PadLeft to length 20 with '\*': " + padLeft);

// PadRight

string padRight = original.PadRight(20, '\*');

Console.WriteLine("PadRight to length 20 with '\*': " + padRight);

// Format

string formatted = string.Format("Formatted: {0} - {1}", "Hello", "World");

Console.WriteLine(formatted);

// Compare

int comparison = string.Compare("abc", "ABC", StringComparison.OrdinalIgnoreCase);

Console.WriteLine("Compare 'abc' and 'ABC' (ignore case): " + comparison);

// Equals

bool equals = original.Equals(" Hello, World! ", StringComparison.Ordinal);

Console.WriteLine("Equals ' Hello, World! ': " + equals);

}

}

using System;

class Program

{

static void Main()

{

// Create an array of strings

string[] colors = { "Red", "Green", "Blue" };

// Join the elements of the array with a comma and space

string result = string.Join(", ", colors);

// Print the result to the console

Console.WriteLine(result);

}

}

//ouput

Red, Green, Blue

**Array**

using System;

using System.Linq;

class Program

{

static void Main()

{

// Initialize an array

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Array Length

Console.WriteLine("Array Length: " + numbers.Length);

// Array IndexOf

int index = Array.IndexOf(numbers, 5);

Console.WriteLine("Index of 5: " + index);

// Array LastIndexOf

int lastIndex = Array.LastIndexOf(numbers, 5);

Console.WriteLine("Last Index of 5: " + lastIndex);

// Array Reverse

Array.Reverse(numbers);

Console.WriteLine("Reversed Array: " + string.Join(", ", numbers));

// Array Sort

Array.Sort(numbers);

Console.WriteLine("Sorted Array: " + string.Join(", ", numbers));

// Array BinarySearch

int searchIndex = Array.BinarySearch(numbers, 5);

Console.WriteLine("Binary Search for 5: " + searchIndex);

// Array Clear

Array.Clear(numbers, 0, 2);

Console.WriteLine("Array after Clear: " + string.Join(", ", numbers));

// Array Copy

int[] copy = new int[numbers.Length];

Array.Copy(numbers, copy, numbers.Length);

Console.WriteLine("Copied Array: " + string.Join(", ", copy));

// Array Clone

int[] clonedArray = (int[])numbers.Clone();

Console.WriteLine("Cloned Array: " + string.Join(", ", clonedArray));

// Array Resize

Array.Resize(ref numbers, 15);

Console.WriteLine("Resized Array: " + string.Join(", ", numbers));

// Array Exists

bool exists = Array.Exists(numbers, element => element == 5);

Console.WriteLine("Does 5 exist in the array? " + exists);

// Array Find

int found = Array.Find(numbers, element => element > 5);

Console.WriteLine("First element greater than 5: " + found);

// Array FindAll

int[] foundAll = Array.FindAll(numbers, element => element > 5);

Console.WriteLine("All elements greater than 5: " + string.Join(", ", foundAll));

// Array FindIndex

int foundIndex = Array.FindIndex(numbers, element => element > 5);

Console.WriteLine("Index of first element greater than 5: " + foundIndex);

// Array FindLast

int foundLast = Array.FindLast(numbers, element => element > 5);

Console.WriteLine("Last element greater than 5: " + foundLast);

// Array FindLastIndex

int foundLastIndex = Array.FindLastIndex(numbers, element => element > 5);

Console.WriteLine("Index of last element greater than 5: " + foundLastIndex);

// Array ForEach

Array.ForEach(numbers, element => Console.Write(element + " "));

Console.WriteLine();

// Array ConvertAll

string[] stringArray = Array.ConvertAll(numbers, element => element.ToString());

Console.WriteLine("Converted to string array: " + string.Join(", ", stringArray));

// Array TrueForAll

bool allPositive = Array.TrueForAll(numbers, element => element >= 0);

Console.WriteLine("Are all elements positive? " + allPositive);

}

}

**List<T>**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

// Initialize a list of integers

List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

// Add

numbers.Add(6);

Console.WriteLine("After Add(6): " + string.Join(", ", numbers));

// AddRange

numbers.AddRange(new List<int> { 7, 8, 9 });

Console.WriteLine("After AddRange(new List<int> { 7, 8, 9 }): " + string.Join(", ", numbers));

// Insert

numbers.Insert(0, 0);

Console.WriteLine("After Insert(0, 0): " + string.Join(", ", numbers));

// InsertRange

numbers.InsertRange(1, new List<int> { -2, -1 });

Console.WriteLine("After InsertRange(1, new List<int> { -2, -1 }): " + string.Join(", ", numbers));

// Remove

numbers.Remove(0);

Console.WriteLine("After Remove(0): " + string.Join(", ", numbers));

// RemoveAt

numbers.RemoveAt(0);

Console.WriteLine("After RemoveAt(0): " + string.Join(", ", numbers));

// RemoveRange

numbers.RemoveRange(0, 2);

Console.WriteLine("After RemoveRange(0, 2): " + string.Join(", ", numbers));

// Contains

bool containsFive = numbers.Contains(5);

Console.WriteLine("Contains 5: " + containsFive);

// IndexOf

int indexOfFive = numbers.IndexOf(5);

Console.WriteLine("Index of 5: " + indexOfFive);

// LastIndexOf

int lastIndexOfFive = numbers.LastIndexOf(5);

Console.WriteLine("Last Index of 5: " + lastIndexOfFive);

// Sort

numbers.Sort();

Console.WriteLine("Sorted List: " + string.Join(", ", numbers));

// Reverse

numbers.Reverse();

Console.WriteLine("Reversed List: " + string.Join(", ", numbers));

// Find

int firstGreaterThanFive = numbers.Find(x => x > 5);

Console.WriteLine("First element greater than 5: " + firstGreaterThanFive);

// FindAll

List<int> allGreaterThanFive = numbers.FindAll(x => x > 5);

Console.WriteLine("All elements greater than 5: " + string.Join(", ", allGreaterThanFive));

// FindIndex

int firstIndexGreaterThanFive = numbers.FindIndex(x => x > 5);

Console.WriteLine("Index of first element greater than 5: " + firstIndexGreaterThanFive);

// FindLast

int lastGreaterThanFive = numbers.FindLast(x => x > 5);

Console.WriteLine("Last element greater than 5: " + lastGreaterThanFive);

// FindLastIndex

int lastIndexGreaterThanFive = numbers.FindLastIndex(x => x > 5);

Console.WriteLine("Index of last element greater than 5: " + lastIndexGreaterThanFive);

// ForEach

Console.Write("Elements: ");

numbers.ForEach(x => Console.Write(x + " "));

Console.WriteLine();

// ConvertAll

List<string> stringList = numbers.ConvertAll(x => x.ToString());

Console.WriteLine("Converted to string list: " + string.Join(", ", stringList));

// TrueForAll

bool allPositive = numbers.TrueForAll(x => x > 0);

Console.WriteLine("Are all elements positive? " + allPositive);

// Count

Console.WriteLine("Count: " + numbers.Count);

// Capacity

Console.WriteLine("Capacity: " + numbers.Capacity);

// Clear

numbers.Clear();

Console.WriteLine("After Clear(): " + string.Join(", ", numbers));

// ToArray

numbers.AddRange(new List<int> { 1, 2, 3, 4, 5 });

int[] array = numbers.ToArray();

Console.WriteLine("ToArray: " + string.Join(", ", array));

}

}

**Dictionary<TKey,TValue>**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

// Initialize a dictionary

Dictionary<string, int> dict = new Dictionary<string, int>();

// Add

dict.Add("One", 1);

dict.Add("Two", 2);

dict.Add("Three", 3);

Console.WriteLine("After Add: " + string.Join(", ", dict));

// ContainsKey

bool containsKey = dict.ContainsKey("One");

Console.WriteLine("ContainsKey(\"One\"): " + containsKey);

// ContainsValue

bool containsValue = dict.ContainsValue(3);

Console.WriteLine("ContainsValue(3): " + containsValue);

// Remove

dict.Remove("Two");

Console.WriteLine("After Remove: " + string.Join(", ", dict));

// TryGetValue

int value;

bool found = dict.TryGetValue("Three", out value);

if (found)

{

Console.WriteLine("Value for key 'Three': " + value);

}

// Keys

Console.WriteLine("Keys: " + string.Join(", ", dict.Keys));

// Values

Console.WriteLine("Values: " + string.Join(", ", dict.Values));

// Count

Console.WriteLine("Count: " + dict.Count);

// Clear

dict.Clear();

Console.WriteLine("After Clear: " + string.Join(", ", dict));

// Using initializer

Dictionary<string, string> countries = new Dictionary<string, string>

{

{ "IN", "India" },

{ "US", "United States" },

{ "GB", "United Kingdom" }

};

Console.WriteLine("Countries: " + string.Join(", ", countries));

}

}

**IEnumerable**

using System;

using System.Collections.Generic;

using System.Linq;

class Program

{

static void Main()

{

// Initialize a list

List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Select

IEnumerable<int> squares = numbers.Select(x => x \* x);

Console.WriteLine("Squares: " + string.Join(", ", squares));

// Where

IEnumerable<int> evenNumbers = numbers.Where(x => x % 2 == 0);

Console.WriteLine("Even Numbers: " + string.Join(", ", evenNumbers));

// Any

bool anyGreaterThanTen = numbers.Any(x => x > 10);

Console.WriteLine("Any greater than 10: " + anyGreaterThanTen);

// All

bool allLessThanTen = numbers.All(x => x < 10);

Console.WriteLine("All less than 10: " + allLessThanTen);

// Count

int count = numbers.Count();

Console.WriteLine("Count: " + count);

// Sum

int sum = numbers.Sum();

Console.WriteLine("Sum: " + sum);

// Min

int min = numbers.Min();

Console.WriteLine("Min: " + min);

// Max

int max = numbers.Max();

Console.WriteLine("Max: " + max);

// Average

double average = numbers.Average();

Console.WriteLine("Average: " + average);

// First

int first = numbers.First();

Console.WriteLine("First: " + first);

// Last

int last = numbers.Last();

Console.WriteLine("Last: " + last);

// Skip

IEnumerable<int> skipTwo = numbers.Skip(2);

Console.WriteLine("Skip 2: " + string.Join(", ", skipTwo));

// Take

IEnumerable<int> takeThree = numbers.Take(3);

Console.WriteLine("Take 3: " + string.Join(", ", takeThree));

// Concat

IEnumerable<int> concatNumbers = numbers.Concat(new List<int> { 11, 12, 13 });

Console.WriteLine("Concatenated Numbers: " + string.Join(", ", concatNumbers));

// Distinct

IEnumerable<int> distinctNumbers = numbers.Concat(new List<int> { 1, 2, 3, 4, 5 });

Console.WriteLine("Distinct Numbers: " + string.Join(", ", distinctNumbers.Distinct()));

}

}

**IEnumerator**

using System;

using System.Collections;

using System.Collections.Generic;

class Program

{

static void Main()

{

// Create a list

List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

// Get the enumerator

IEnumerator<int> enumerator = numbers.GetEnumerator();

// MoveNext and Current

Console.WriteLine("Using MoveNext and Current:");

while (enumerator.MoveNext())

{

int current = enumerator.Current;

Console.WriteLine(current);

}

// Reset

enumerator.Reset();

// Using foreach (which internally uses IEnumerator)

Console.WriteLine("Using foreach:");

foreach (int number in numbers)

{

Console.WriteLine(number);

}

// Using non-generic IEnumerator

Console.WriteLine("Using non-generic IEnumerator:");

IEnumerator nonGenericEnumerator = numbers.GetEnumerator();

while (nonGenericEnumerator.MoveNext())

{

int current = (int)nonGenericEnumerator.Current;

Console.WriteLine(current);

}

}

}

**IQueryable**

using System;

using System.Collections.Generic;

using System.Linq;

class Program

{

static void Main()

{

// Create a list

List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

// Create an IQueryable from the list

IQueryable<int> queryableNumbers = numbers.AsQueryable();

// Where

IQueryable<int> evenNumbersQuery = queryableNumbers.Where(x => x % 2 == 0);

Console.WriteLine("Even Numbers:");

foreach (int number in evenNumbersQuery)

{

Console.WriteLine(number);

}

// Select

IQueryable<string> stringNumbersQuery = queryableNumbers.Select(x => x.ToString());

Console.WriteLine("Numbers as Strings:");

foreach (string number in stringNumbersQuery)

{

Console.WriteLine(number);

}

// OrderBy

IQueryable<int> orderedNumbersQuery = queryableNumbers.OrderBy(x => x);

Console.WriteLine("Ordered Numbers:");

foreach (int number in orderedNumbersQuery)

{

Console.WriteLine(number);

}

// Skip and Take

IQueryable<int> skipAndTakeQuery = queryableNumbers.Skip(2).Take(2);

Console.WriteLine("Skip 2, Take 2:");

foreach (int number in skipAndTakeQuery)

{

Console.WriteLine(number);

}

// Any

bool anyGreaterThanFive = queryableNumbers.Any(x => x > 5);

Console.WriteLine("Any greater than 5: " + anyGreaterThanFive);

// All

bool allLessThanTen = queryableNumbers.All(x => x < 10);

Console.WriteLine("All less than 10: " + allLessThanTen);

// Count

int count = queryableNumbers.Count();

Console.WriteLine("Count: " + count);

// Sum

int sum = queryableNumbers.Sum();

Console.WriteLine("Sum: " + sum);

// Min

int min = queryableNumbers.Min();

Console.WriteLine("Min: " + min);

// Max

int max = queryableNumbers.Max();

Console.WriteLine("Max: " + max);

// Average

double average = queryableNumbers.Average();

Console.WriteLine("Average: " + average);

}

}

**Delegates**

using System;

// Declare a delegate

delegate int Operation(int a, int b);

class Program

{

static void Main()

{

// Assign a method to the delegate

Operation add = Add;

Operation subtract = Subtract;

// Use the delegate

Console.WriteLine("Addition: " + PerformOperation(5, 3, add));

Console.WriteLine("Subtraction: " + PerformOperation(5, 3, subtract));

}

// Method to add two numbers

static int Add(int a, int b)

{

return a + b;

}

// Method to subtract two numbers

static int Subtract(int a, int b)

{

return a - b;

}

// Method that takes a delegate as a parameter

static int PerformOperation(int a, int b, Operation operation)

{

return operation(a, b);

}

}

**Delegates Enhancements**

using System;

// Declare a delegate

delegate void GreetingDelegate(string message);

class Program

{

static void Main()

{

// Simplified delegate invocation

GreetingDelegate greet = message => Console.WriteLine($"Hello, {message}!");

greet("world");

// Target-typed new expressions for delegates

GreetingDelegate greet2 = new(message => Console.WriteLine($"Hello again, {message}!"));

greet2("C#");

// Lambda expression enhancements

Func<int, int, int> add = (int a, int b) => a + b;

Console.WriteLine($"3 + 5 = {add(3, 5)}");

Action<string> printMessage = message =>

{

Console.WriteLine("Message:");

Console.WriteLine(message);

};

printMessage("Hello, C#!");

// Combine delegates

GreetingDelegate combined = greet + greet2;

combined("combined delegates");

}

}

**Extension Methods**

using System;

// Define a static class for extension methods

public static class StringExtensions

{

// Define an extension method for the string class

public static string Reverse(this string input)

{

char[] charArray = input.ToCharArray();

Array.Reverse(charArray);

return new string(charArray);

}

}

class Program

{

static void Main()

{

string original = "hello";

string reversed = original.Reverse();

Console.WriteLine("Original: " + original);

Console.WriteLine("Reversed: " + reversed);

// Using LINQ extension methods

int[] numbers = { 1, 2, 3, 4, 5 };

int sum = numbers.Sum();

Console.WriteLine("Sum of numbers: " + sum);

}

}

**LINQ - Language Integrated Query**

using System;

using System.Collections.Generic;

using System.Linq;

class Program

{

static void Main()

{

// Sample data

List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// 1. Where: filters a sequence of values based on a predicate

var filteredNumbers = numbers.Where(n => n % 2 == 0);

Console.WriteLine("Filtered Numbers:");

foreach (var number in filteredNumbers)

{

Console.WriteLine(number);

}

// 2. Select: projects each element of a sequence into a new form

var squaredNumbers = numbers.Select(n => n \* n);

Console.WriteLine("\nSquared Numbers:");

foreach (var number in squaredNumbers)

{

Console.WriteLine(number);

}

// 3. OrderBy: sorts the elements of a sequence in ascending order

var orderedNumbers = numbers.OrderBy(n => n);

Console.WriteLine("\nOrdered Numbers:");

foreach (var number in orderedNumbers)

{

Console.WriteLine(number);

}

// 4. OrderByDescending: sorts the elements of a sequence in descending order

var descendingNumbers = numbers.OrderByDescending(n => n);

Console.WriteLine("\nDescending Numbers:");

foreach (var number in descendingNumbers)

{

Console.WriteLine(number);

}

// 5. Any: determines whether any element of a sequence satisfies a condition

bool anyNumbersGreaterThan10 = numbers.Any(n => n > 10);

Console.WriteLine($"\nAny Numbers Greater Than 10: {anyNumbersGreaterThan10}");

// 6. All: determines whether all elements of a sequence satisfy a condition

bool allNumbersLessThan20 = numbers.All(n => n < 20);

Console.WriteLine($"\nAll Numbers Less Than 20: {allNumbersLessThan20}");

// 7. Contains: determines whether a sequence contains a specified element

bool containsNumber5 = numbers.Contains(5);

Console.WriteLine($"\nContains Number 5: {containsNumber5}");

// 8. Count: returns the number of elements in a sequence

var count = numbers.Count();

Console.WriteLine($"\nCount of Numbers: {count}");

// 9. Sum: computes the sum of a sequence of numeric values

var sum = numbers.Sum();

Console.WriteLine($"\nSum of Numbers: {sum}");

// 10. Min: returns the minimum value in a sequence of numeric values

var min = numbers.Min();

Console.WriteLine($"\nMinimum Number: {min}");

// 11. Max: returns the maximum value in a sequence of numeric values

var max = numbers.Max();

Console.WriteLine($"\nMaximum Number: {max}");

// 12. Average: computes the average of a sequence of numeric values

var average = numbers.Average();

Console.WriteLine($"\nAverage Number: {average}");

// 13. First: returns the first element of a sequence

var first = numbers.First();

Console.WriteLine($"\nFirst Number: {first}");

// 14. Last: returns the last element of a sequence

var last = numbers.Last();

Console.WriteLine($"\nLast Number: {last}");

// 15. ElementAt: returns the element at a specified index in a sequence

var elementAt = numbers.ElementAt(3);

Console.WriteLine($"\nElement At Index 3: {elementAt}");

// 16. Concat: concatenates two sequences

List<int> moreNumbers = new List<int> { 11, 12, 13, 14, 15 };

var concatenatedNumbers = numbers.Concat(moreNumbers);

Console.WriteLine("\nConcatenated Numbers:");

foreach (var number in concatenatedNumbers)

{

Console.WriteLine(number);

}

// 17. Distinct: returns distinct elements from a sequence

List<int> duplicateNumbers = new List<int> { 1, 1, 2, 2, 3, 3, 4, 4, 5, 5 };

var distinctNumbers = duplicateNumbers.Distinct();

Console.WriteLine("\nDistinct Numbers:");

foreach (var number in distinctNumbers)

{

Console.WriteLine(number);

}

// 18. Except: returns the difference between two sequences

var exceptNumbers = numbers.Except(moreNumbers);

Console.WriteLine("\nNumbers Except More Numbers:");

foreach (var number in exceptNumbers)

{

Console.WriteLine(number);

}

// 19. Intersect: returns the intersection of two sequences

var intersectNumbers = numbers.Intersect(moreNumbers);

Console.WriteLine("\nIntersected Numbers:");

foreach (var number in intersectNumbers)

{

Console.WriteLine(number);

}

// 20. Union: returns the union of two sequences

var unionNumbers = numbers.Union(moreNumbers);

Console.WriteLine("\nUnion of Numbers:");

foreach (var number in unionNumbers)

{

Console.WriteLine(number);

}

}

}

**Ref and Out**

using System;

class Program

{

static void Main()

{

int x = 10;

int y;

// Using ref keyword

Add(ref x, 5);

Console.WriteLine("Value of x after using ref: " + x);

// Using out keyword

Multiply(10, 20, out y);

Console.WriteLine("Value of y after using out: " + y);

}

static void Add(ref int a, int b)

{

a += b;

}

static void Multiply(int a, int b, out int result)

{

result = a \* b;

}

}