

C# Quick Reference

Types, LINQ, async/await, collections, OOP essentials

Basics

Hello World

```
Console.WriteLine("Hello, World!"); // top-level (C# 10+)
// Classic: class Program { static void Main() { ... } }
```

Build & Run

```
dotnet new console -n MyApp # create project
dotnet run # compile and run
dotnet build # compile only
```

Variables & Constants

```
int x = 42;
var name = "Alice"; // type inference
const double Pi = 3.14159;
readonly int maxRetries = 3; // set once, in ctor
```

Types

Value Types

int	32-bit signed integer
long	64-bit signed integer
float	32-bit floating point (suffix f)
double	64-bit floating point
decimal	128-bit high precision (suffix m)
bool	true / false
char	16-bit Unicode character

Reference Types

string	Immutable UTF-16 text
object	Base type for all types
dynamic	Bypasses compile-time type checking
int[]	Array of integers
List<T>	Generic list (System.Collections.Generic)

Nullable & Tuples

```
int? age = null; // nullable value type
string? name = null; // nullable reference (C# 8+)
var point = (X: 1, Y: 2); // named tuple
Console.WriteLine(point.X);
```

String Features

```
string name = "World";
string msg = $"Hello, {name}!"; // interpolation
string path = @"C:\Users\file.txt"; // verbatim
string raw = """raw "string" here"""; // raw (C# 11+)
```

Control Flow

If / Else

```
if (x > 0) Console.WriteLine("positive");
else if (x == 0) Console.WriteLine("zero");
else Console.WriteLine("negative");
```

Switch & Pattern Matching

```
string label = x switch {
    > 0 => "positive", 0 => "zero", _ => "negative"
};
if (obj is string s && s.Length > 0) { } // pattern match
```

Loops

```
for (int i = 0; i < 10; i++) { }
foreach (var item in collection) { }
while (condition) { }
do { } while (condition);
```

Classes

Class Definition

```
public class Person {
    public string Name { get; set; }
    public int Age { get; init; } // init-only (C# 9+)
    public Person(string name, int age) { Name = name; Age = age; }
}
```

Records (C# 9+)

```
public record Point(double X, double Y);
var p1 = new Point(1, 2);
var p2 = p1 with { X = 3 }; // non-destructive copy
// auto: Equals, GetHashCode, ToString, deconstruct
```

Inheritance

```
public abstract class Shape { public abstract double Area(); }
public class Circle(double r) : Shape {
    public override double Area() => Math.PI * r * r;
}
```

Access Modifiers

public	Accessible from anywhere
private	Same class only (default for members)
protected	Same class and derived classes
internal	Same assembly only (default for classes)
protected internal	Same assembly or derived classes

Interfaces

Interface Definition

```
public interface IShape {
    double Area();
    double Perimeter() => 0; // default impl (C# 8+)
}
public class Rect(double w, double h) : IShape { public double
Area() => w * h; }
```

Common Interfaces

IEnumerable<T>	Iteration support (foreach, LINQ)
IDisposable	Deterministic cleanup (using statement)
IComparable<T>	Natural ordering for sorting
IEquatable<T>	Value equality comparison
ICloneable	Object cloning

LINQ

Method Syntax

```
var result = numbers
    .Where(n => n > 3)
    .OrderBy(n => n)
    .Select(n => n * 2)
    .ToList();
```

Query Syntax

```
var result = from n in numbers
              where n > 3
              orderby n
              select n * 2;
```

Common LINQ Methods

.Where(pred)	Filter elements
.Select(func)	Project / transform elements
.OrderBy(key)	Sort ascending
.GroupBy(key)	Group elements by key
.First() / .FirstOrDefault()	First element (or default)
.Any(pred)	true if any element matches
.Count()	Number of elements
.Sum() / .Average()	Aggregate numeric values
.Distinct()	Remove duplicates
.SelectMany(func)	Flatten nested collections

Async/Await

Async Method

```
public async Task<string> FetchAsync(string url) {
    using var client = new HttpClient();
    return await client.GetStringAsync(url);
}
```

Task Combinators

```
var results = await Task.WhenAll(task1, task2, task3);
var first = await Task.WhenAny(task1, task2);
```

Async Patterns

Task	Async void return (no result)
Task<T>	Async return with result of type T
ValueTask<T>	Lightweight task for sync-fast paths
await foreach	Async iteration over IAsyncEnumerable<T>
CancellationToken	Cooperative cancellation for async ops

Collections

Common Collections

List<T>	Dynamic array, fast index access
Dictionary<K, V>	Hash map, O(1) lookup by key
HashSet<T>	Unique elements, O(1) lookup
Queue<T>	FIFO collection
Stack<T>	LIFO collection
LinkedList<T>	Doubly-linked list
SortedDictionary<K, V>	Sorted by key (tree-based)

Dictionary Usage

```
var dict = new Dictionary<string, int> {
    ["Alice"] = 90, ["Bob"] = 85
};
dict.TryGetValue("Alice", out int score);
foreach (var (key, val) in dict) { }
```

Immutable Collections

```
using System.Collections.Immutable;
var list = ImmutableList.Create(1, 2, 3);
var newList = list.Add(4); // returns new list
```

Properties

Property Syntax

```
public string Name { get; set; }
public int Age { get; private set; }
public string Email { get; init; } // init-only
public string Display => $"{Name} ({Age})"; // computed
```

C# Quick Reference

Indexers

```
public double this[int row, int col] {  
    get => data[row, col];  
    set => data[row, col] = value;  
}
```

Property Patterns

{ get; set; }	Read-write auto-property
{ get; }	Read-only (set in constructor only)
{ get; init; }	Read-only after initialization (C# 9+)
{ get; private set; }	Publicly readable, privately writable
=> expression	Expression-bodied (computed) property

Exceptions

Try / Catch / Finally

```
try { int result = int.Parse(input); }  
catch (FormatException ex) { Console.Error.WriteLine(ex.Message); }  
catch (Exception ex) when (ex is not OutOfMemoryException) { }  
finally { /* always executes */ }
```

Using Statement

```
using var file = File.OpenRead("data.txt");  
// file.Dispose() called automatically at scope end  
// equivalent to try/finally with Dispose()
```

Common Exceptions

ArgumentNullException	Null argument passed to method
ArgumentOutOfRangeException	Argument outside valid range
InvalidOperationException	Operation invalid for current state
NullReferenceException	Dereference of null object
KeyNotFoundException	Key not found in dictionary
NotImplementedException	Method not yet implemented

Custom Exception

```
public class AppException : Exception {  
    public int Code { get; }  
    public AppException(string msg, int code)  
        : base(msg) { Code = code; }  
}
```