C# 빠른 참조

타입, LINQ, async/await, 컬렉션, OOP 핵심

기본

Hello World

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

빌드 & 실행

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

변수 & 상수

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

타입

값 타입

`int`	32비트 부호 있는 정수
`long`	64비트 부호 있는 정수
`float`	32비트 부동소수점 (접미사 `f`)
`double`	64비트 부동소수점
`decimal`	128비트 고정밀도 (접미사 `m`)
`bool`	`true` / `false`
`char`	16비트 유니코드 문자

참조 타입

`string`	불변 UTF-16 텍스트
`object`	모든 타입의 기반 타입
`dynamic`	컴파일 타임 타입 검사 우회
`int[]`	정수 배열
`List<T>`	제네릭 리스트 (System.Collections.Generic)

nullable & 튜플

```
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);
```

문자열 기능

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

제어 흐름

If / Else

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

Switch & 패턴 매칭

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

반복문

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

클래스

클래스 정의

```
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; }
}
```

레코드 (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
```

상속

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

접근 제한자

`public`	어디서든 접근 가능
`private`	동일 클래스만 (멤버 기본값)
`protected`	동일 클래스 및 파생 클래스
`internal`	동일 어셈블리만 (클래스 기본값)
`protected internal`	동일 어셈블리 또는 파생 클래스

인터페이스

인터페이스 정의

```
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; }
```

주요 인터페이스

`IEnumerable<T>`	순회 지원 (foreach, LINQ)
`IDisposable`	결정론적 정리 (`using` 구문)
`IComparable<T>`	정렬을 위한 자연 순서
`IEquatable<T>`	값 동등성 비교
`ICloneable`	객체 복제

LINQ

메서드 문법

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

쿼리 문법

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

주요 LINQ 메서드

`.Where(pred)`	요소 필터링
`.Select(func)`	요소 투영 / 변환
`.OrderBy(key)`	오름차순 정렬
`.GroupBy(key)`	키로 요소 그룹화
`.First()` / `.FirstOrDefault()`	첫 번째 요소 (또는 기본값)
`.Any(pred)`	매칭 요소가 있으면 `true`
`.Count()`	요소 수
`.Sum()` / `.Average()`	수치 집계
`.Distinct()`	중복 제거
`.SelectMany(func)`	중첩 컬렉션 평탄화

Async/Await

비동기 메서드

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

Task 조합

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

비동기 패턴

`Task`	비동기 void 반환 (결과 없음)
`Task<T>`	T 타입 결과를 반환하는 비동기
`ValueTask<T>`	동기-빠른 경로를 위한 경량 태스크
`await foreach`	`IAsyncEnumerable<T>` 비동기 순회
`CancellationToken`	비동기 작업의 협력적 취소

컬렉션

주요 컬렉션

`List<T>`	동적 배열, 빠른 인덱스 접근
`Dictionary<K,V>`	해시 맵, O(1) 키 조회
`HashSet<T>`	고유 요소, O(1) 조회
`Queue<T>`	FIFO 컬렉션
`Stack<T>`	LIFO 컬렉션
`LinkedList<T>`	이중 연결 리스트
`SortedDictionary<K,V>`	키로 정렬 (트리 기반)

Dictionary 사용

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

불변 컬렉션

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

프로퍼티

프로퍼티 문법

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

인덱서

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

프로퍼티 패턴

`{ get; set; }`	읽기-쓰기 자동 프로퍼티
`{ get; }`	읽기 전용 (생성자에서만 설정)
`{ get; init; }`	초기화 후 읽기 전용 (C# 9+)
`{ get; private set; }`	공개 읽기, 비공개 쓰기
`=> expression`	식 본문 (계산) 프로퍼티

예외

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 구문

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

주요 예외

`ArgumentNullException`	null 인수가 메서드에 전달됨
`ArgumentOutOfRangeException`	유효 범위 밖의 인수
`InvalidOperationException`	현재 상태에서 유효하지 않은 작업
`NullReferenceException`	null 객체 역참조
`KeyNotFoundException`	딕셔너리에서 키를 찾을 수 없음
`NotImplementedException`	아직 구현되지 않은 메서드

커스텀 예외

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