C# クイックリファレンス

型、LINQ、async/await、コレクション、OOP 必須事項

基礎

Hello World

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

ビルド & 実行

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

変数 & 定数

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

型

値型

`int`	32 ビット符号付き整数
`long`	64 ビット符号付き整数
`float`	32 ビット浮動小数点（サフィックス `f`）
`double`	64 ビット浮動小数点
`decimal`	128 ビット高精度（サフィックス `m`）
`bool`	`true` / `false`
`char`	16 ビット Unicode 文字

参照型

`string`	不変の UTF-16 テキスト
`object`	全型の基底型
`dynamic`	コンパイル時の型チェックを省略
`int[]`	整数の配列
`List<T>`	ジェネリックリスト（System.Collections.Generic）

Nullable & タプル

```
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);
```

文字列機能

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

制御フロー

If / Else

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

Switch & パターンマッチング

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

ループ

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

クラス

クラスの定義

```
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; }
}
```

レコード（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
```

継承

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

アクセス修飾子

`public`	どこからでもアクセス可能
`private`	同一クラスのみ（メンバーのデフォルト）
`protected`	同一クラスおよび派生クラス
`internal`	同一アセンブリのみ（クラスのデフォルト）
`protected internal`	同一アセンブリまたは派生クラス

インターフェイス

インターフェイスの定義

```
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; }
```

よく使うインターフェイス

`IEnumerable<T>`	イテレーションサポート（foreach、LINQ）
`IDisposable`	確定的クリーンアップ（`using` 文）
`IComparable<T>`	ソート用の自然順序
`IEquatable<T>`	値の等値比較
`ICloneable`	オブジェクトのクローン

LINQ

メソッド構文

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

クエリ構文

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

よく使う LINQ メソッド

`.Where(pred)`	要素をフィルタリング
`.Select(func)`	要素を射影 / 変換
`.OrderBy(key)`	昇順にソート
`.GroupBy(key)`	キーで要素をグループ化
`.First() / .FirstOrDefault()`	最初の要素（またはデフォルト）
`.Any(pred)`	いずれかの要素がマッチすれば `true`
`.Count()`	要素数
`.Sum() / .Average()`	数値の集約
`.Distinct()`	重複を削除
`.SelectMany(func)`	ネストされたコレクションをフラット化

Async/Await

非同期メソッド

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

タスクコンビネータ

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

非同期パターン

`Task`	非同期の void 戻り値（結果なし）
`Task<T>`	型 T の結果を持つ非同期戻り値
`ValueTask<T>`	同期が速いパス向けの軽量タスク
`await foreach`	`IAsyncEnumerable<T>` の非同期イテレーション
`CancellationToken`	非同期操作の協調的キャンセル

コレクション

よく使うコレクション

`List<T>`	動的配列、高速なインデックスアクセス
`Dictionary<K,V>`	ハッシュマップ、O(1) のキールックアップ
`HashSet<T>`	ユニーク要素、O(1) ルックアップ
`Queue<T>`	FIFO コレクション
`Stack<T>`	LIFO コレクション
`LinkedList<T>`	双方向リンクリスト
`SortedDictionary<K,V>`	キーでソート済み（ツリーベース）

Dictionary の使い方

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

イミュータブルコレクション

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

プロパティ

プロパティ構文

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

インデクサー

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

プロパティパターン

`{ get; set; }`	読み書き可能な自動プロパティ
`{ get; }`	読み取り専用（コンストラクタでのみ設定）
`{ get; init; }`	初期化後は読み取り専用（C# 9+）
`{ get; private set; }`	パブリック読み取り、プライベート書き込み
`=> expression`	式本体（計算済み）プロパティ

例外

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 文

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

よく使う例外

`ArgumentNullException`	メソッドに null 引数が渡された
`ArgumentOutOfRangeException`	引数が有効範囲外
`InvalidOperationException`	現在の状態では無効な操作
`NullReferenceException`	null オブジェクトの逆参照
`KeyNotFoundException`	辞書にキーが見つからない
`NotImplementedException`	メソッドが未実装

カスタム例外

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