

Referência Rápida C#

Tipos, LINQ, async/await, coleções, essenciais de OOP

Básico

Hello World

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

Build e Execução

```
dotnet new console -n MyApp # criar projeto
dotnet run # compilar e executar
dotnet build # somente compilar
```

Variáveis e Constantes

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

Tipos

Tipos por Valor

int	Inteiro de 32 bits com sinal
long	Inteiro de 64 bits com sinal
float	Ponto flutuante de 32 bits (sufixo f)
double	Ponto flutuante de 64 bits
decimal	Alta precisão de 128 bits (sufixo m)
bool	true / false
char	Caractere Unicode de 16 bits

Tipos por Referência

string	Texto UTF-16 imutável
object	Tipo base para todos os tipos
dynamic	Ignora verificação de tipos em tempo de compilação
int[]	Array de inteiros
List<T>	Lista genérica (System.Collections.Generic)

Nullable e Tuplas

```
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);
```

Recursos de String

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

Controle de Fluxo

If / Else

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

Switch e Pattern Matching

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

Laços

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

Classes

Definição de Classe

```
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
```

Herança

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

Modificadores de Acesso

public	Acessível de qualquer lugar
private	Somente na mesma classe (padrão para membros)
protected	Mesma classe e classes derivadas
internal	Somente no mesmo assembly (padrão para classes)
protected internal	Mesmo assembly ou classes derivadas

Interfaces

Definição de Interface

```
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; }
```

Interfaces Comuns

IEnumerable<T>	Suporte à iteração (foreach, LINQ)
IDisposable	Limpeza determinística (instrução using)
IComparable<T>	Ordenação natural para classificação
IEquatable<T>	Comparação de igualdade por valor
ICloneable	Clonagem de objetos

LINQ

Sintaxe de Método

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

Sintaxe de Query

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

Métodos LINQ Comuns

.Where(pred)	Filtrar elementos
.Select(func)	Projetar / transformar elementos
.OrderBy(key)	Ordenar em ordem crescente
.GroupBy(key)	Agrupar elementos por chave
.First() / .FirstOrDefault()	Primeiro elemento (ou padrão)
.Any(pred)	true se algum elemento corresponder
.Count()	Número de elementos
.Sum() / .Average()	Agregar valores numéricos
.Distinct()	Remover duplicatas
.SelectMany(func)	Nivelar coleções aninhadas

Async/Await

Método Async

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

Combinadores de Task

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

Padrões Async

Task	Retorno void assíncrono (sem resultado)
Task<T>	Retorno assíncrono com resultado do tipo T
ValueTask<T>	Task leve para caminhos rápidos síncronos
await foreach	Iteração assíncrona sobre IAsyncEnumerable<T>
CancellationToken	Cancelamento cooperativo para operações assíncronas

Coleções

Coleções Comuns

List<T>	Array dinâmico, acesso por índice rápido
Dictionary<K,V>	Hash map, busca O(1) por chave
HashSet<T>	Elementos únicos, busca O(1)
Queue<T>	Coleção FIFO
Stack<T>	Coleção LIFO
LinkedList<T>	Lista duplamente encadeada
SortedDictionary<K,V>	Ordenado por chave (baseado em árvore)

Uso de Dictionary

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

Coleções Imutáveis

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

Propriedades

Sintaxe de Propriedade

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

Referência Rápida C#

Indexadores

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

Padrões de Propriedade

{ get; set; }	Auto-propriedade de leitura e escrita
{ get; }	Somente leitura (definir apenas no construtor)
{ get; init; }	Somente leitura após inicialização (C# 9+)
{ get; private set; }	Leitura pública, escrita privada
=> expression	Propriedade com corpo de expressão (computada)

Exceções

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 */ }
```

Instrução Using

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

Exceções Comuns

ArgumentNullException	Argumento nulo passado ao método
ArgumentOutOfRangeException	Argumento fora do intervalo válido
InvalidOperationException	Operação inválida para o estado atual
NullReferenceException	Desreferência de objeto nulo
KeyNotFoundException	Chave não encontrada no dicionário
NotImplementedException	Método ainda não implementado

Exceção Personalizada

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