

Riferimento Rapido C#

Tipi, LINQ, async/await, collezioni, essenziali OOP

Basi

Hello World

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

Build ed Esecuzione

```
dotnet new console -n MyApp # crea progetto
dotnet run # compila ed esegui
dotnet build # solo compilazione
```

Variabili e Costanti

```
int x = 42;
var name = "Alice"; // inferenza di tipo
const double Pi = 3.14159;
readonly int maxRetries = 3; // impostato una volta, nel costruttore
```

Tipi

Tipi Valore

int	Intero con segno a 32 bit
long	Intero con segno a 64 bit
float	Virgola mobile a 32 bit (suffisso f)
double	Virgola mobile a 64 bit
decimal	Alta precisione 128 bit (suffisso m)
bool	true / false
char	Carattere Unicode a 16 bit

Tipi Riferimento

string	Testo UTF-16 immutabile
object	Tipo base per tutti i tipi
dynamic	Bypassa il controllo di tipo a compile-time
int[]	Array di interi
List<T>	Lista generica (System.Collections.Generic)

Nullable e Tuple

```
int? age = null; // tipo valore nullable
string? name = null; // riferimento nullable (C# 8+)
var point = (X: 1, Y: 2); // tupla nominata
Console.WriteLine(point.X);
```

Funzionalità Stringa

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

Controllo del Flusso

If / Else

```
if (x > 0) Console.WriteLine("positivo");
else if (x == 0) Console.WriteLine("zero");
else Console.WriteLine("negativo");
```

Switch e Pattern Matching

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

Cicli

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

Classi

Definizione di 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; }
}
```

Record (C# 9+)

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

Ereditarietà

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

Modificatori di Accesso

public	Accessibile da ovunque
private	Solo stessa classe (default per i membri)
protected	Stessa classe e classi derivate
internal	Solo stesso assembly (default per le classi)
protected internal	Stesso assembly o classi derivate

Interfacce

Definizione di Interfaccia

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

Interfacce Comuni

IEnumerable<T>	Supporto iterazione (foreach, LINQ)
IDisposable	Pulizia deterministica (istruzione using)
IComparable<T>	Ordinamento naturale per la classificazione
IComparable<T>	Confronto di uguaglianza per valore
ICloneable	Clonazione oggetti

LINQ

Sintassi a Metodo

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

Sintassi a Query

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

Metodi LINQ Comuni

.Where(pred)	Filtra gli elementi
.Select(func)	Proietta/trasforma gli elementi
.OrderBy(key)	Ordina in modo crescente
.GroupBy(key)	Raggruppa gli elementi per chiave
.First() / .FirstOrDefault()	Primo elemento (o default)
.Any(pred)	true se qualsiasi elemento corrisponde
.Count()	Numero di elementi
.Sum() / .Average()	Aggregazione valori numerici
.Distinct()	Rimuove i duplicati
.SelectMany(func)	Appiattisce le collezioni annidate

Async/Await

Metodo Async

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

Combinatori Task

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

Pattern Async

Task	Ritorno async void (nessun risultato)
Task<T>	Ritorno async con risultato di tipo T
ValueTask<T>	Task leggero per percorsi sincroni veloci
await foreach	Iterazione async su IAsyncEnumerable<T>
CancellationToken	Cancellazione cooperativa per operazioni async

Collezioni

Collezioni Comuni

List<T>	Array dinamico, accesso per indice veloce
Dictionary<K, V>	Hash map, ricerca O(1) per chiave
HashSet<T>	Elementi unici, ricerca O(1)
Queue<T>	Collezione FIFO
Stack<T>	Collezione LIFO
LinkedList<T>	Lista doppiamente concatenata
SortedDictionary<K, V>	Ordinato per chiave (basato su albero)

Utilizzo Dictionary

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

Collezioni Immutabili

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

Proprietà

Sintassi delle Proprietà

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

Riferimento Rapido C#

Indicizzatori

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

Pattern Proprietà

{ get; set; }	Auto-proprietà lettura-scrittura
{ get; }	Sola lettura (impostabile solo nel costruttore)
{ get; init; }	Sola lettura dopo inizializzazione (C# 9+)
{ get; private set; }	Pubblicamente leggibile, privatamente scrivibile
=> expression	Proprietà expression-bodied (calcolata)

Eccezioni

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 { /* eseguito sempre */ }
```

Istruzione Using

```
using var file = File.OpenRead("data.txt");  
// file.Dispose() chiamato automaticamente allo scope end  
// equivalente a try/finally con Dispose()
```

Eccezioni Comuni

ArgumentNullException	Argomento null passato al metodo
ArgumentOutOfRangeException	Argomento fuori dall'intervallo valido
InvalidOperationException	Operazione non valida per lo stato corrente
NullReferenceException	Dereferenzamento di oggetto null
KeyNotFoundException	Chiave non trovata nel dizionario
NotSupportedException	Metodo non ancora implementato

Eccezione Personalizzata

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