C# - 1

Slide 1 – Introduzione a C#

C# è un **linguaggio orientato agli oggetti** sviluppato da Microsoft, parte dell'ecosistema **.NET**.

Nasce per essere potente ma leggibile, unendo efficienza e sicurezza del tipo statico.

```
// Programma base in C#
using System;

class Program {
    static void Main() {
        Console.WriteLine("Ciao, mondo!");
    }
}
```

Slide 2 – Struttura di un programma

Un programma C# è composto da **classi**, **metodi** e **namespace**.

Il metodo `Main()` è il **punto d'ingresso** del programma.

```
namespace Esempio {
    class Program {
        static void Main() {
            Console.WriteLine("Esecuzione iniziata");
        }
    }
}
```

Slide 3 – Tipi di dato

Ogni variabile ha un **tipo statico**. I tipi più comuni sono: `int`, `float`, `double`, `bool`, `string`, `char`.

I tipi **value** vengono memorizzati nello stack, i **reference** nell'heap.

```
int numero = 10;
string nome = "Marius";
bool attivo = true;
```

Slide 4 – Variabili e costanti

Le variabili memorizzano dati modificabili, le costanti restano fisse.

Si usano le parole chiave `var` o `const` per inferenza e valori immutabili.

```
var messaggio = "Hello!";
const double PI = 3.1415;
```

Slide 5 – Operatori

Gli **operatori** permettono di eseguire operazioni matematiche, logiche e di confronto.

Esempi: `+`, `-`, `*`, `/`, `%`, `==`, `!=`, `&&`, `||`.

```
int a = 5, b = 2;
bool confronto = a > b; // true
int somma = a + b; // 7
```

Slide 6 – Strutture condizionali

`if`, `else if`, `else` gestiscono scelte nel codice.

Valutano **espressioni booleane** e decidono quale blocco eseguire.

```
int eta = 18;
if (eta >= 18) Console.WriteLine("Maggiorenne");
else Console.WriteLine("Minorenne");
```

Slide 7 – Switch

`switch` è utile per confrontare una variabile con più valori possibili.

Dal C# 8 è possibile usare **switch expression** più compatte.

```
string giorno = "Lunedì";
switch (giorno) {
    case "Lunedì": Console.WriteLine("Inizio settimana"); break;
    case "Venerdì": Console.WriteLine("Quasi weekend"); break;
    default: Console.WriteLine("Giorno qualunque"); break;
}
```

Slide 8 – Ciclo for

`for` è usato per ripetere istruzioni un numero noto di volte.

Comprende **inizializzazione**, **condizione** e **incremento**.

```
for (int i = 0; i < 5; i++) {
    Console.WriteLine(i);
}
```

Slide 9 – Ciclo while e do-while

`while` ripete finché la condizione è vera; `do-while` esegue almeno una volta.

Sono usati quando non si conosce a priori il numero di iterazioni.

```
int n = 0;
while (n < 3) {
    Console.WriteLine(n);
    n++;
}
```

Slide 10 – Foreach

`foreach` itera su collezioni o array senza gestire indici manualmente.

È sicuro e leggibile, ma non permette modifiche dirette all'elemento.

```
string[] nomi = {"Anna", "Luca", "Marius"};
foreach (string nome in nomi) {
    Console.WriteLine(nome);
}
```

Slide 11 – Array

Un **array** contiene più valori dello stesso tipo.

Ha dimensione fissa e si accede con l'indice a partire da 0.

```
int[] numeri = {1, 2, 3};
Console.WriteLine(numeri[1]); // 2
```

Slide 12 – Liste

Le **liste** (`List<T>`) sono dinamiche e fanno parte di `System.Collections.Generic`.

Possono aggiungere o rimuovere elementi facilmente.

```
using System.Collections.Generic;

List<string> nomi = new List<string>() {"Anna", "Luca"};
nomi.Add("Marius");
Console.WriteLine(nomi.Count); // 3
```

Slide 13 – Metodi

I **metodi** contengono logica riutilizzabile.

Possono avere parametri e restituire valori con `return`.

```
int Somma(int a, int b) {
    return a + b;
}

Console.WriteLine(Somma(3, 5)); // 8
```

Slide 14 – Overloading dei metodi

L'**overloading** consente di definire più metodi con lo stesso nome ma parametri diversi.

Il compilatore sceglie il più adatto in base ai tipi passati.

```
void Stampa(string testo) ⇒ Console.WriteLine(testo);
void Stampa(int numero) ⇒ Console.WriteLine($"Numero: {numero}");
```

Slide 15 – Classi e oggetti

Le **classi** definiscono modelli; gli **oggetti** sono istanze reali di quelle classi.

Ogni classe può avere campi, metodi e costruttori.

```
class Persona {
    public string Nome;
    public void Saluta() ⇒ Console.WriteLine($"Ciao, sono {Nome}");
}

Persona p = new Persona();
p.Nome = "Marius";
p.Saluta();
```

Slide 16 – Costruttori

Un **costruttore** inizializza automaticamente gli oggetti quando vengono creati.

Può ricevere parametri per impostare valori iniziali.

```
class Persona {
    public string Nome;
    public Persona(string nome) {
        Nome = nome;
    }
}
var p = new Persona("Marius");
```

Slide 17 – Incapsulamento e proprietà

L'**incapsulamento** protegge i dati interni.

Si usano **proprietà** (`get` e `set`) per controllare l'accesso ai campi.

```
class Persona {
    private int eta;
    public int Eta {
        get ⇒ eta;
        set { if (value > 0) eta = value; }
    }
}
```

Slide 18 – Ereditarietà

L'**ereditarietà** permette a una classe di estendere un'altra.

La sottoclasse eredita metodi e attributi della classe base.

```
class Animale {
    public void Dormi() ⇒ Console.WriteLine("Zzz");
}

class Cane : Animale {
    public void Abbaia() ⇒ Console.WriteLine("Bau!");
}
```

Slide 19 – Polimorfismo

Il **polimorfismo** consente a metodi con lo stesso nome di comportarsi diversamente.

Si ottiene con `virtual` e `override`.

```
class Animale { public virtual void Verso() ⇒ Console.WriteLine("Suono gen
erico"); }
class Gatto : Animale { public override void Verso() ⇒ Console.WriteLine
("Miao"); }
```

Slide 20 – Gestione delle eccezioni

Le **eccezioni** permettono di gestire errori senza interrompere il programma.

`try-catch-finally` è la struttura standard di controllo.

```
try {
    int x = 5 / 0;
}
catch (Exception e) {
    Console.WriteLine("Errore: " + e.Message);
}
finally {
    Console.WriteLine("Fine esecuzione");
}
```