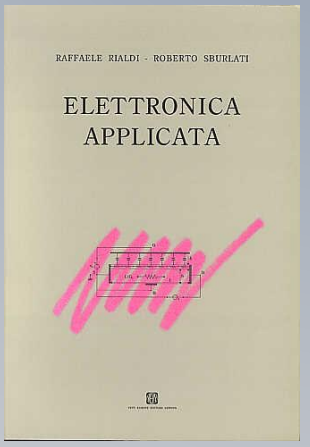
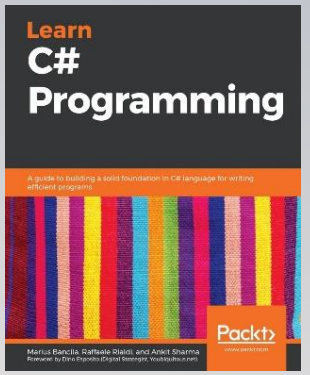
Le novità di .NET 9

Ing. Raffaele Rialdi
Senior Software Architect - Consultant
@raffaeler - raffaeler@vevy.com

Chi sono

- Laurea Master in Ingegneria Elettronica (Unige)
- Lavoro professionalmente nel software dal 1987
- Insegno saltuariamente a Ingegneria Informatica (Unige)
- Membro della commissione ICT dell'Ordine degli Ingegneri
- Microsoft Most Valuable Professional per 21 anni consecutivi
- Libero professionista, Software Architect in diversi ambiti:
  - Financial, Manufacturing, Healthcare, F1 racing, ...
- Speaker in conferenze nel mondo (più di 200 interventi in 20 anni)
  - Europa, Asia, USA
- Co-Autore del libro "Elettronica Applicata"
- Co-Autore del libro "C# Programming"
- Presidente di DotNetLiguria a Genova





@raffaeler

.NET 9

- C# 13 introduce piccole ma importanti novità
  - Codice più compatto
  - Nuove funzionalità che migliorano le performance
- .NET Runtime: notevoli miglioramenti di prestazioni
  - Compilatore JIT
    - Massive ottimizzazioni per ARM64
  - Garbage Collector
    - Il Server GC ora prova sempre a ridurre l'utilizzo di memoria. Utilissimo per i container!
  - Allocazione di oggetti nello stack
- Librerie: tantissime novità "invisibili"
  - Write Barriers
  - Nuove strategie di Inlining.

Un esempio per tutti: Performance di Linq

Method	Runtime	Mean	Ratio	Allocated
Chunk	.NET 8.0	10.7213 ns	1.00	72 B
Chunk	.NET 9.0	4.1320 ns	0.39	–
Distinct	.NET 8.0	9.4410 ns	1.00	64 B
Distinct	.NET 9.0	0.7162 ns	0.08	–
GroupJoin	.NET 8.0	22.4746 ns	1.00	144 B
GroupJoin	.NET 9.0	1.1356 ns	0.05	–
Join	.NET 8.0	18.6332 ns	1.00	168 B
Join	.NET 9.0	1.3585 ns	0.07	–
ToLookup	.NET 8.0	23.3518 ns	1.00	128 B
ToLookup	.NET 9.0	0.9539 ns	0.04	–
Reverse	.NET 8.0	9.5791 ns	1.00	48 B
Reverse	.NET 9.0	0.9947 ns	0.10	–

Method	Runtime	Mean	Ratio	Allocated
SelectIndex	.NET 8.0	11.1235 ns	1.00	72 B
SelectIndex	.NET 9.0	0.5603 ns	0.05	–
SelectMany	.NET 8.0	10.7537 ns	1.00	64 B
SelectMany	.NET 9.0	0.9906 ns	0.09	–
SkipWhile	.NET 8.0	11.2900 ns	1.00	72 B
SkipWhile	.NET 9.0	1.0988 ns	0.10	–
TakeWhile	.NET 8.0	11.8818 ns	1.00	72 B
TakeWhile	.NET 9.0	1.0381 ns	0.09	–
WhereIndex	.NET 8.0	11.1751 ns	1.00	80 B
WhereIndex	.NET 9.0	1.2185 ns	0.11	–
SequenceEqual	.NET 8.0	26'623.3ns	1.00	–
SequenceEqual	.NET 9.0	913.4ns	0.03	–

https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-9/#linq

Le novità di C# 13

Usare «params» sulle collection

- Tutti conosciamo params

```
void DoSomething(params int[] numbers) { }
```

- Non era inusuale creare un overload con IEnumerable<T> o IEnumerable

```
void DoSomething(IEnumerable<int> numbers) { }
```

- Ora è stato esteso alle collection

```
void DoSomething(params IEnumerable<int> numbers)
```

- Ed anche a ReadOnlySpan<T> e Span<T>

```
void DoSomething(params ReadOnlySpan<int> numbers)
```

Il nuovo tipo di lock chiamato “Lock”

- La nuova classe System.Threading.Lock migliora le prestazioni della Critical Section
- La regione di codice tra “Enter” ed “Exit” può essere access da un solo thread alla volta
- Lo statement “lock” di C# è stato modificato per poter usare Lock

Method	Mean
FillOld1	17.76 ns
FillOld2	17.14 ns
FillNew1	14.72 ns
FillNew2	14.90 ns
FillNew3	13.96 ns

```
private object _oldLock = new();

lock (_oldLock)
{
    // ...
}
```

```
private Lock _newLock = new();

lock (_newLock)
{
    // ...
}
```

Proprietà e indexer parziali

- Dopo i partial methods, anche le proprietà possono essere ‘parziali’
- Utilissimi per la generazione di codice

```
public partial class SomeClass
{
    public partial int Number
        { get; set; }

    public partial int this[int i]
        { get; set; }
}
```

```
public partial class SomeClass
{
    private int _number;
    public partial int Number
    {
        get => _number;
        set { _number = value; }
    }

    public partial int this[int i]
    {
        get => _number;
        set { _number = value; }
    }
}
```

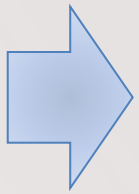
Solo in preview: la nuova keyword “field”

- La nuova keyword field referenzia il campo usato nelle auto properties

```
private int _y;
public int Y
{
    get => _y;
    set => _y = value
;
}
```

- Equivale a

```
public int X
{
    get => field;
    set => field = value
;
}
```



```
[CompilerGenerated]
[DebuggerBrowsable(DebuggerBrowsableState.Never)]
private int <X>k__BackingField;

private int _y;

public int X
{
    get
    {
        return <X>k__BackingField;
    }
    set
    {
        <X>k__BackingField = value;
    }
}
```

Disassemblato con ILSpy

Le novità nella class library

Nuova classe per le conversion Base64

- La classe Convert esegue la conversione standard che è inadatta agli URL
- Invece di rimpiazzare i caratteri invalidi ("+"e "/") si può usare Base64Url

- Encode/Decode di: !"#$%&'()*+,-./0123456789:;<=>?

Classe	Encode	Decode
Convert	ISIjJCUmJygpKissLS4vMDEyMzQ1Njc4OTo7PD0+Pw==	!"#$%&'()*+,-./0123456789:;<=>?
Base64Url	ISIjJCUmJygpKissLS4vMDEyMzQ1Njc4OTo7PD0-Pw	!"#$%&'()*+,-./0123456789:;<=>?

Nuove classi degne di nota

- ReadOnlySet<T>
- X509CertificateLoader
- System.Diagnostics.Metrics.Gauge<T>
  - Una metrica per valori non additivi
- PersistedAssemblyBuilder
  - Salvare tipi e metadati creati dinamicamente
- TypeName (package System.Reflection.Metadata)
  - Un parser di tipi ECMA-335 senza necessità di caricare il tipo in memoria
- Tensor<T> (System.Numeric.Tensors)

Il nuovo Task.WhenEach

- Conosciamo già:
  - Task.WaitAny e Task.WaitAll (da evitare perché bloccanti)
  - Task.WhenAny e Task.WhenAll
- Con .NET 9 è stato introdotto Task.WhenEach
  - Ritorna un IAsyncEnumerable<Task<T>>

```
Task<string> t1 = Do(slow);
Task<string> t2 = Do(fast);
Task<string> t3 = Do(medium);

await foreach (var t in Task.WhenEach(t1, t2, t3))
{
    Console.WriteLine(t.Result);
}
```

Linq: i nuovi CountBy e AggregateBy

```
var numbers = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var count = numbers.CountBy(n => n % 2 == 0);
foreach (var c in count)
{
    Console.WriteLine(
        $"CountBy> {c.Key}:{c.Value}");
}
```

CountBy> False:5
CountBy> True:5

```
var aggregate = numbers.AggregateBy(n => n % 2 == 0,
    seed:0, (acc, n) => acc + n);
foreach (var c in aggregate)
{
    Console.WriteLine(
        $"AggregateBy> {c.Key}:{c.Value}");
}
```

AggregateBy> False:25
AggregateBy> True:30

Altri nuovi metodi interessanti

- I nuovi overload di DateTime e TimeSpan
  - I metodi From* con argomento intero per una maggiore precisione
- La classe Activity (Telemetria) ha il nuovo metodo AddLink
  - Permette di linkare una Activity ad un'altra

```
var activity = new Activity("LinkTest");
activity.AddLink(new ActivityLink(activityContext););
```

- Le novità in «SearchValues»
- Websocket keep-alive e timeout
- Guid.CreateVersion7
  - Guid ordinabili con il sort ma randomici.

Interoperabilità in-process con Python

Il progetto CSnakes

- È il progetto di un dipendente Microsoft che lavora dentro il team Python
  - https://github.com/tonybaloney/CSnakes
    - Il nome Tony Baloney è un alias derivato da uno scherzo noto in UK
- Il progetto giusto al momento giusto
  - Riceve contribuzioni da personaggi di spicco in Microsoft
    - David Fowler, Tim Heuer, Aaron Robinson, Jared Parsons, …
- Permette di generare il codice di interoperabilità necessario a invocare Python in-process dentro un processo .NET
- Supporta i virtual environment di Python ("venv")
- Funziona cross-platform.

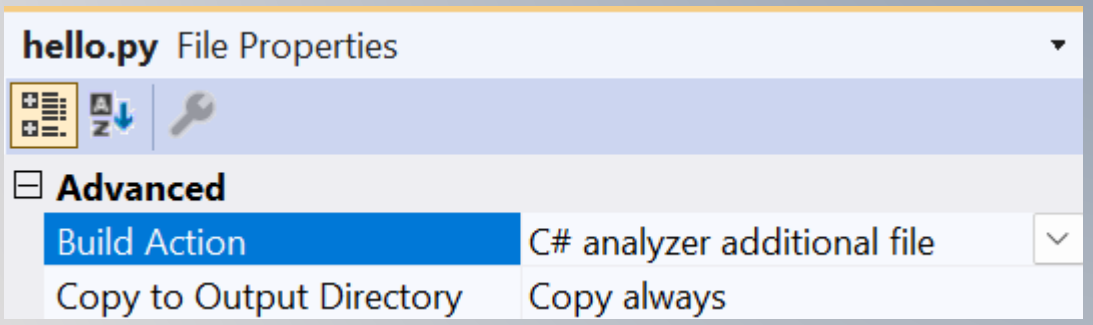
Si parte dalla Dependency Injection di .NET

- Includere il pacchetto NuGet "CSnakes.Runtime"
  - Sia nella applicazione Host che nelle eventuali DLL con listati Python
- Configurazione della Dependency Injection:
  - var pythonBuilder = services.WithPython();

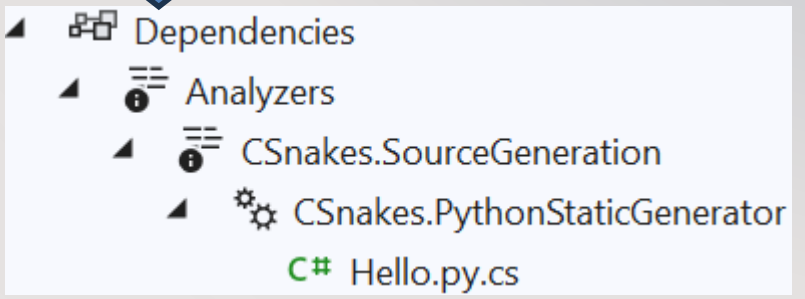
```
pythonBuilder
    .WithHome(home) // path ai sorgenti Python
    .WithVirtualEnvironment(venv) // path al virtual env
    .FromFolder(userPython, "3.13"); // path a Python.exe
    .WithPipInstaller();
services.AddSingleton(p => p.GetRequiredService<IPythonEnvironment>().Hello())
;

// Alternative a .FromFolder:
    .FromWindowsInstaller("3.13")
    .FromNuGet("3.13.0")
    .FromMacOSInstallerLocator("3.13")
    .FromEnvironmentVariable("Python3_ROOT_DIR", "3.13")
```

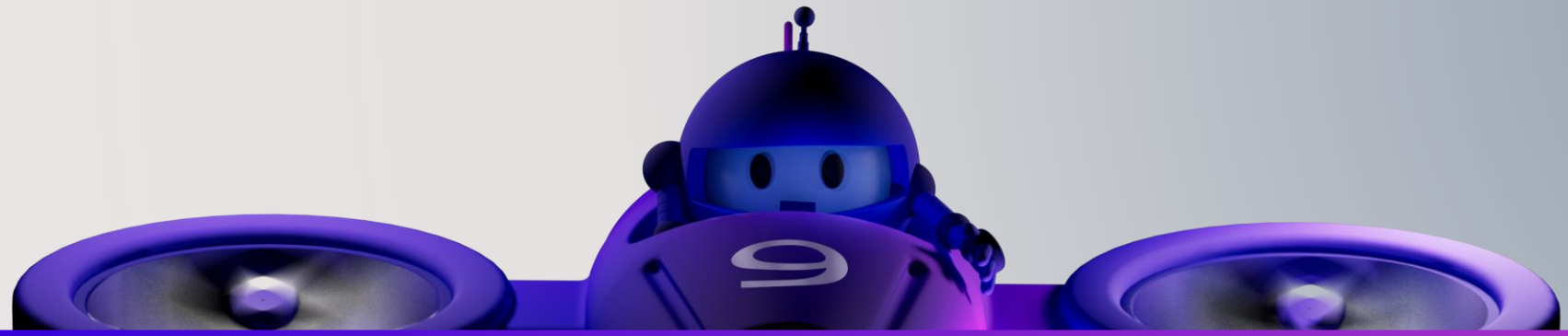
Il generatore di interoperabilità Python

- Il sorgente Python viene aggiunto come "AdditionalFile"
- Il generatore crea immediatamente il codice di interoperabilità
- La classe "Hello" è ora disponibile nel "PythonEnvironment"
- La classe "Hello" a sua volta può essere registrata in Dependency Injection.




```
<ItemGroup>
  <AdditionalFiles Include="hello.py">
    <CopyToOutputDirectory>Always
    </CopyToOutputDirectory>
  </AdditionalFiles>
</ItemGroup>
```





Get .NET 9

Download .NET 9
aka.ms/get-dotnet-9

Compilate i Feedback
Grazie!