C# COMO VOCÊ (TALVEZ) NUNCA VIU

CÓDIGO LIMPO, EXPRESSIVO,
SÓLIDO & FUNCIONAL.

ELEMAR JR
falecom@elemarjr.com
elemarjr@ravendb.net
elemarjr.com

CÓDIGO PROFISSIONAL

Alta performance ou código para um domínio complexo?

Obter performance ou explicitar o domínio no código é uma tarefa difícil. Sou um desenvolvedor experiente, focado em resultados, que mantem alto padrão de qualidade.

ARQUITETURA DE SOFTWARE

Nuvem? Escalabilidade? Performance? Legado? Novo software?

É necessário pensar cuidadosamente nos componentes, responsabilidades e relacionamentos de sua solução. Sou um solucionador de problemas criativo e posso ajudar.

ESTRATÉGIA & INOVAÇÃO

Preocupado em desenvolver, mais que bons produtos, ótimos negócios?

A inovação é a ferramenta fundamental do empreendedor. Sem bons resultados não há inovação! Há duas décadas desenvolvo negócios de classe mundial.

RAVENDB

Safe by Default, Optimized for Efficiency

@ayende


Entre em contato conosco: +55 (11) 4111-1353
Solicite uma demonstração
Acompanhe-nos!
guiando
Controle, visibilidade
e redução de custos.
HOME
A GUIANDO
SOLUÇÕES
CLIENTES
CONTEÚDO
CONTATO
Controle total de custos
de forma prática e eficiente!
Reduza custos com telecom, impressão,
licenças de software, leasing e facilities
Um software de eficiência e redução de custos
Clique e se surpreenda

ARE YOU A
PROFESSIONAL?


Abstrato
Concreto


EXPRESSIVIDADE

```
public static int SumAllEvens(int[] numbers)
{
    var accum = 0;
    for (var i = 0; i < numbers.Length; i++)
    {
        if (numbers[i] % 2 == 0)
        {
            accum += numbers[i];
        }
    }
    return accum;
}
```

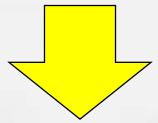
Abstrato

```
public static IEnumerable<int> AllEvens(int[] numbers)
{
    [redacted]
}

public static int Sum(IEnumerable<int> numbers)
{
    [redacted]
}

public static int SumAllEvens(int[] number)
{
    return Sum(AllEvens(number));
}
```

Abstrato

```
public static int SumAllEvens(int[] numbers)
{
    return numbers
        .Where(n => n % 2 == 0)
        .Sum();
}
```

```
public static int SumAllEvens(int[] numbers)
{
    return numbers
        .Where(n => n % 2 == 0)
        .Sum();
}
```



```
public static int SumAllEvens(int[] numbers) => numbers
        .Where(n => n % 2 == 0)
        .Sum();
```

IMUTABILIDADE

```
public interface IShape
{
    double SideA { get; set; }
    double SideB { get; set; }
    bool CanRotate { get; set; }
    double Area { get; set; }
}
```

```
sealed class Shape
{
    public Shape(double sideA, double sideB)
    {
        SideA = sideA;
        SideB = sideB;
    }

    public double SideA { get; }
    public double SideB { get; }

    public Shape WithSideA(double newSideA) =>
        new Shape(newSideA, SideB);

    public Shape WithSideB(double newSideB) =>
        new Shape(SideA, newSideB);
}
```

```
public sealed class CuttingPlan
{
    public IEnumerable<Shape> ShapesToCut { get; }
    public IPlacementStrategy PlacementStrategy { get; }
    public double Margin { get; }
    public CuttingPlan(IEnumerable<Shape> shapesToCut, IPlacementStrategy placementStrategy, double margin) : this(
        shapesToCut, placementStrategy, margin, new CuttingLayout[0])
    { }
    public CuttingPlan(Enumerable<Shape> shapesToCut, IPlacementStrategy placementStrategy, double margin,
        IEnumerable<CuttingLayout> layouts)
    {
        ShapesToCut = shapesToCut;
        PlacementStrategy = placementStrategy;
        Margin = margin;
        Layouts = layouts;
    }
    public IEnumerable<CuttingLayout> Layouts { get; }
    public bool IsCompleted => !ShapesToCut.Any();
    public CuttingPlan AddLayout(CuttingLayout layout)
    {
        var newShapesToCut =
            ShapesToCut.Where(shape => layout.Placements.All(p => p.Shape.Concrete() != shape)).ToList();
        return new CuttingPlan(newShapesToCut, PlacementStrategy, Margin, Layouts.Concat(new[] { layout }));
    }
}
```

```
public sealed class CuttingPlan
{
    public IEnumerable<Shape> ShapesToCut { get; }
    public IPlacementStrategy PlacementStrategy { get; }
    public double Margin { get; }
    public Cu                                          oCut, IPlacementStrategy placementStrategy, double margin) : this(
        shape                                          ew CuttingLayout[0])
    { }
    public Cu                                          Cut, IPlacementStrategy placementStrategy, double margin,
        IEnum
    {
        Shape
        Place
        Margin = margin;
        Layouts = layouts;
    }
    public IEnumerable<Cuttin      out> Layouts { get; }
    public bool IsCompleted => !ShapesToCut.Any();
    public CuttingPlan AddLayout(CuttingLayout layout)
    {
        var newShapesToCut =
            ShapesToCut.Where(shape => layout.Placements.All(p => p.Shape.Concrete() != shape)).ToList();
        return new CuttingPlan(newShapesToCut, PlacementStrategy, Margin, Layouts.Concat(new[] { layout }));
    }
}
```

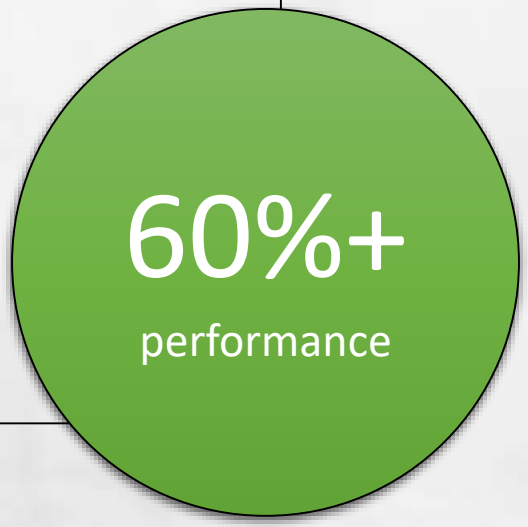
Pure Function

Sem side-effects, para os mesmos parâmetros retorna sempre o mesmo resultado.

(Thread Safe)

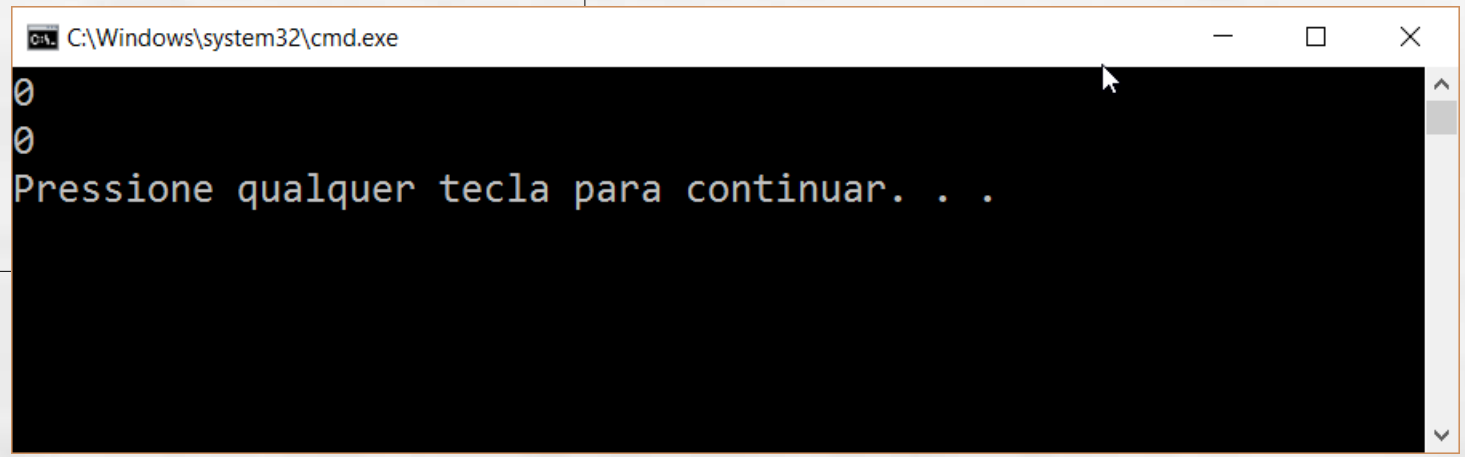
```
protected override void ApplyMutations(
    IList<Hint> arrayToMutate,
    double mutationRate,
    int ignore)
{
    for (var i = ignore; i < arrayToMutate.Count; i++)
    {
        arrayToMutate[i] = arrayToMutate[i].Mutate(mutationRate);
    }
}
protected override void FillWithCrossovers(IList<Hint> arrayToFill)
{
    for (var i = 0; i < arrayToFill.Count; i++)
    {
        if (arrayToFill[i] != null) continue;
        var parent1 = PickUsingTournament();
        var parent2 = PickUsingTournament();
        arrayToFill[i] = parent1.CrossOver(parent2);
    }
}
```

```
protected override void ApplyMutations(C#
    IList<Hint> arrayToMutate,
    double mutationRate,
    int ignore)
{
    Parallel.For(ignore, arrayToMutate.Count, (i) =>
    {
        arrayToMutate[i] = arrayToMutate[i].Mutate(mutationRate);
    });
}
protected override void FillWithCrossovers(IList<Hint> arrayToFill)
{
    Parallel.For(0, arrayToFill.Count, (i) =>
    {
        if (arrayToFill[i] != null) return;
        var parent1 = PickUsingTournament();
        var parent2 = PickUsingTournament();
        arrayToFill[i] = parent1.CrossOver(parent2);
    });
}
```



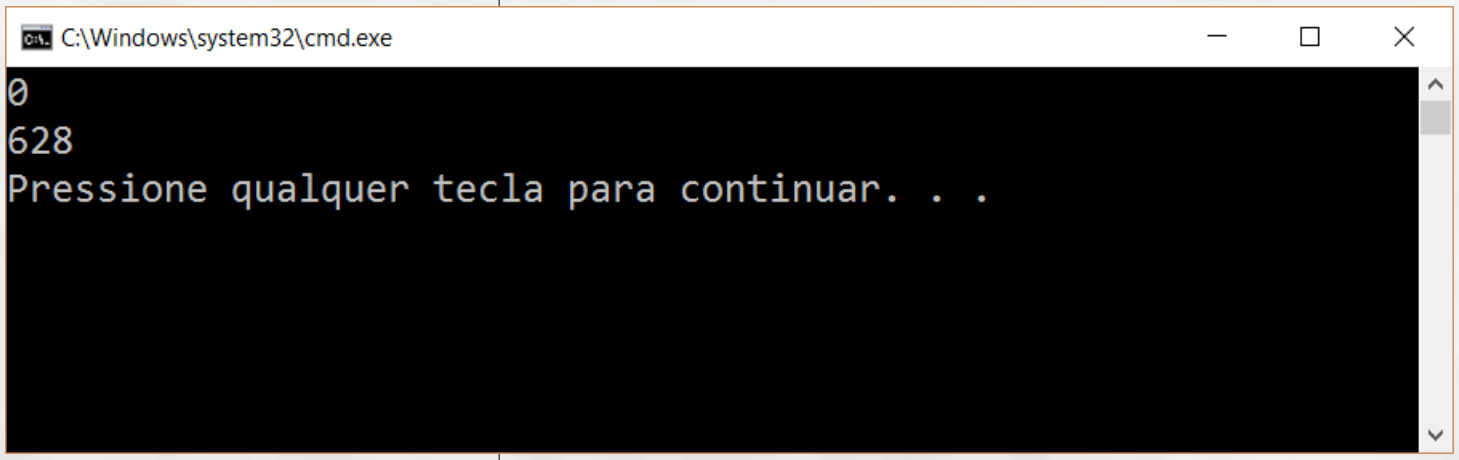
```
using System.Linq;

using static System.Linq.Enumerable;
using static System.Console;

class Program
{
    static void Main()
    {
        var numbers = Range(start: -10000, count: 20001)
            .Reverse()
            .ToList(); // 10000 ... 0 ... -10000

        WriteLine(numbers.Sum());

        numbers.Sort();
        WriteLine(numbers.Sum());

    }
}
```

```
using System.Linq;

using static System.Linq.Enumerable;
using static System.Console;

class Program
{
    static void Main()
    {
        var numbers = Range(start: -10000, count: 20001)
            .Reverse()
            .ToList(); // 10000 ... 0 ... -10000

        WriteLine(numbers.Sum());

        numbers.Sort();
        WriteLine(numbers.Sum());

    }
}
```



```
using System;
using System.Linq;
using System.Threading.Tasks;

using static System.Linq.Enumerable;
using static System.Console;

class Program
{
    static void Main()
    {
        var numbers = Range(start: -10000, count: 20001)
            .Reverse()
            .ToList(); // 10000 ... -10000


        Action task1 = () => WriteLine(numbers.Sum());
        Action task2 = () =>
        {
            numbers.Sort();
            WriteLine(numbers.Sum());
        };

        Parallel.Invoke(task1, task2);
    }
}
```

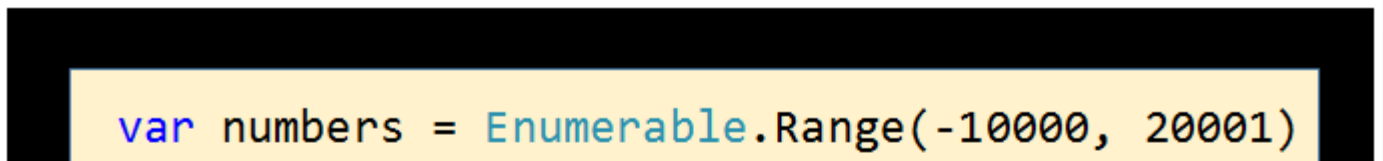
```
using System;
using System.Linq;
using System.Threading.Tasks;

using static System.Linq.Enumerable;
using static System.Console;

class Program
{
    static void Main()
    {
        var numbers = Range(start: -10000, count: 20001)
            .Reverse()
            .ToList(); // 10000 ... -10000


        Action task1 = () => WriteLine(numbers.Sum());
        Action task2 = () =>
        {
            numbers.Sort();
            WriteLine(numbers.Sum());
        };

        Parallel.Invoke(task1, task2);
    }
}
```



```
using System;
using System.Linq;
using System.Threading.Tasks;

using static System.Linq.Enumerable;
using static System.Console;

class Program
{
    static void Main()
    {
        var numbers = Range(start: -10000, count: 20001)
            .Reverse()
            .ToList(); // 10000 ... -10000


        Action task1 = () => WriteLine(numbers.Sum());
        Action task2 = () =>
        {
            numbers.Sort();
            WriteLine(numbers.Sum());
        };

        Parallel.Invoke(task1, task2);
    }
}
```


```
C:\Windows\system32\cmd.exe
0
628
Pressione qualquer tecla para continuar. . .
```

```
var numbers = Enumerable.Range(-10000, 20001)
                        .Reverse()
                        .ToList();
```



```
Console.WriteLine(numbers.Sum());
```

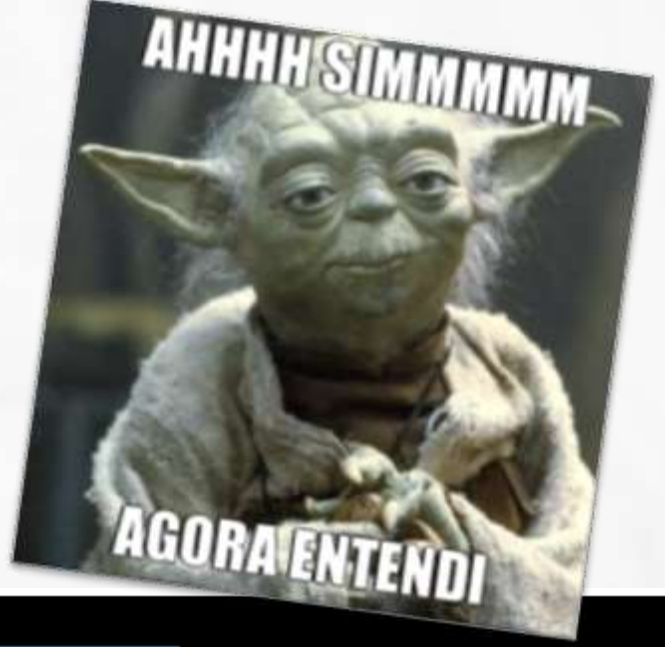
```
numbers.Sort();
Console.WriteLine(numbers.Sum());
```


NÃO ENTENDI NADA!


```
var numbers = Enumerable.Range(-10000, 20001)
                        .Reverse()
                        .ToList();
```

```
Console.WriteLine(numbers.Sum());
```

```
numbers.Sort();
Console.WriteLine(numbers.Sum());
```



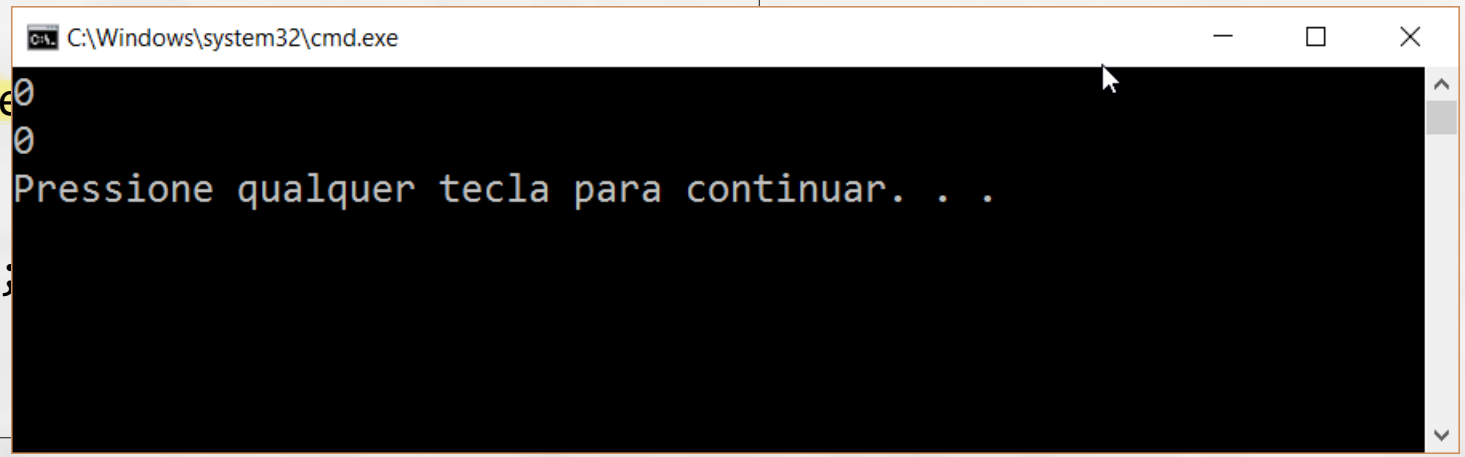

```
var numbers = Enumerable.Range(-10000, 20001)
                        .Reverse()
                        .ToList();
```

```
Console.WriteLine(numbers.Sum());
```

```
numbers.Sort();
Console.WriteLine(numbers.Sum());
```

```
using System;
using System.Linq;
using System.Threading.Tasks;

using static System.Linq.Enumerable;
using static System.Console;
class Program
{
    static void Main()
    {
        var numbers = Range(start: -10000, count: 20001)
            .Reverse()
            .ToList(); // 10000 ... -10000

        Action task1 = () => WriteLine(numbers.Sum());
        Action task2 = () =>
        {
            var ordered = numbers.OrderBy(n => n);
            WriteLine(ordered.Sum());
        };
        Parallel.Invoke(task1, task2);
    }
}
```

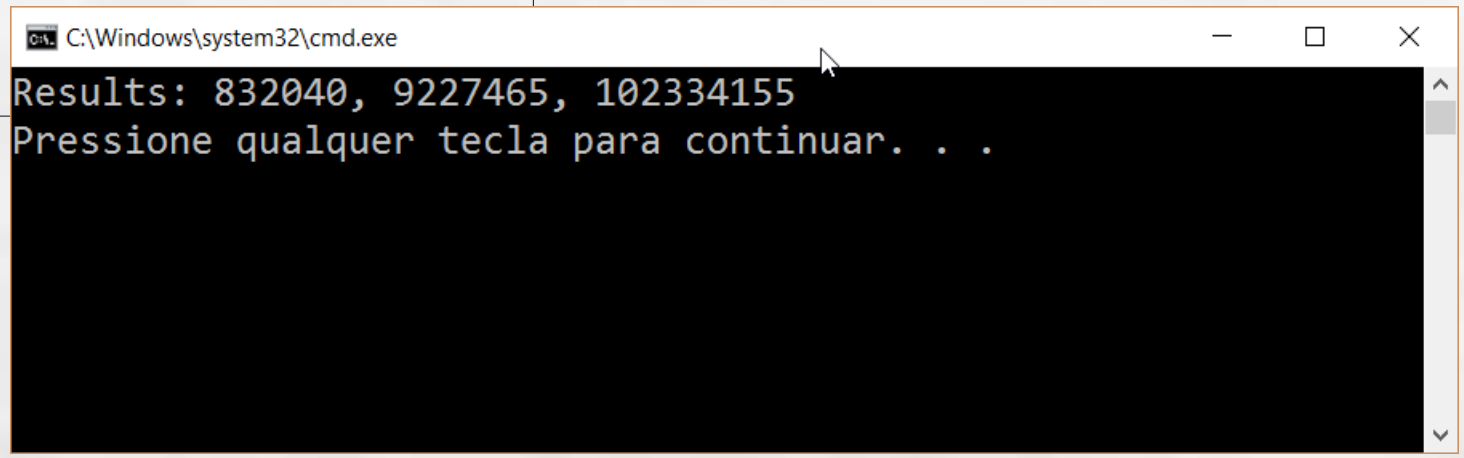
```
using System;
using System.Linq;
using System.Threading.Tasks;

using static System.Linq.Enumerable;
using static System.Console;
class Program
{
    static void Main()
    {
        var numbers = Range(start: -10000, count: 20001)
            .Reverse()
            .ToList(); // 10000 ... -10000

        Action task1 = () => WriteLine(numbers.Sum());
        Action task2 = () =>
        {
            var ordered = numbers.Orde
            WriteLine(ordered.Sum());
        };
        Parallel.Invoke(task1, task2);
    }
}
```



HIGH-ORDER FUNCTIONS

```
using static System.Console;
class Program
{
    static void Main()
    {
        var f30 = Fibonacci(30);
        var f35 = Fibonacci(35);
        var f40 = Fibonacci(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }
    public static int Fibonacci(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
}
```

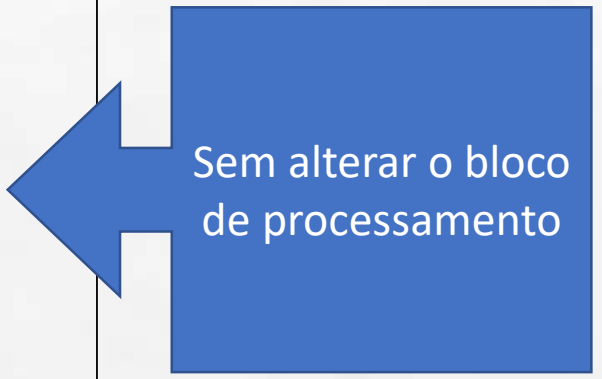
```
using static System.Console;
class Program
{
    static void Main()
    {
        var f30 = Fibonacci(30);
        var f35 = Fibonacci(35);
        var f40 = Fibonacci(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }
    public static int Fibonacci(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
}
```



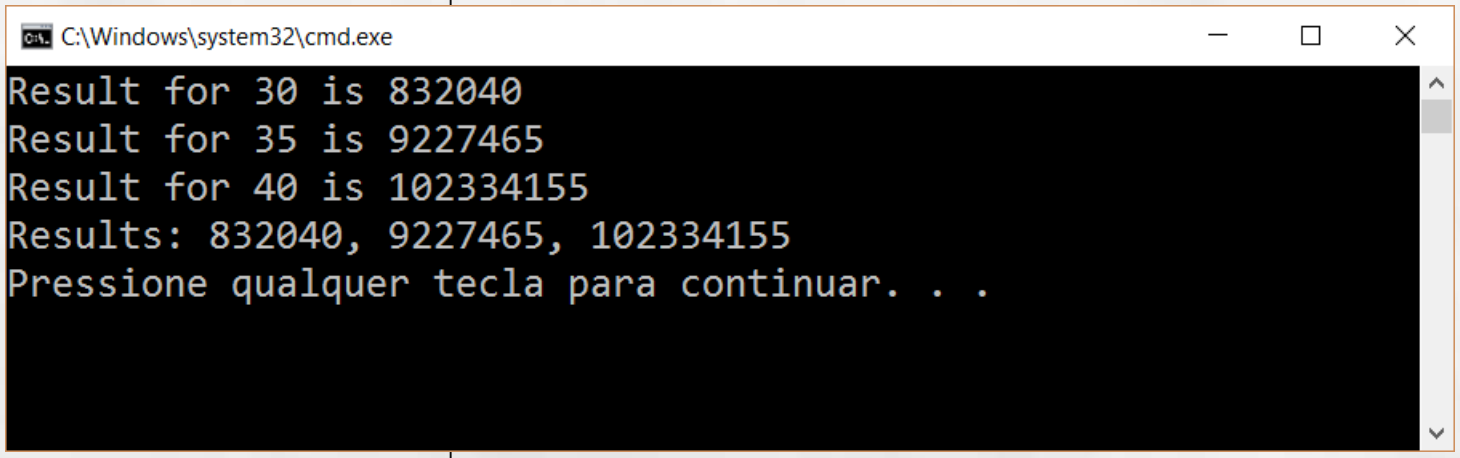
Abstrato

```
using System;
using static System.Console;
class Program
{
    static void Main()
    {
        Func<int, int> fibo = Fibonacci;
        var f30 = fibo(30);
        var f35 = fibo(35);
        var f40 = fibo(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }
    public static int Fibonacci(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
}
```

```
using System;
using static System.Console;
class Program
{
    static void Main()
    {
        Func<int, int> fibo = n =>
        {
            var result = Fibonacci(n);
            WriteLine($"Result for {n} is {result}");
            return result;
        };

        var f30 = fibo(30);
        var f35 = fibo(35);
        var f40 = fibo(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }
    public static int Fibonacci(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
}
```



```
using System;
using static System.Console;
class Program
{
    static void Main()
    {
        Func<int, int> fibo = n =>
        {
            var result = Fibonacci(n);
            WriteLine($"Result for {n} is {result}");
            return result;
        };

        var f30 = fibo(30);
        var f35 = fibo(35);
        var f40 = fibo(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }
    public static int Fibonacci(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + Fibonacci
    }
}
```



```
using static System.Console;
class Program
{
    static void Main()
    {
        int fibo(int n)
        {
            var result = Fibonacci(n);
            WriteLine($"Result for {n} is {result}");
            return result;
        };

        var f30 = fibo(30);
        var f35 = fibo(35);
        var f40 = fibo(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }
    public static int Fibonacci(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
}
```

```
using System;
using static System.Console;
class Program
{
    static void Main()
    {
        int printResultsOf(Func<int, int> func, int n)
        {
            var result = func(n);
            WriteLine($"Result for {n} is {result}");
            return result;
        }

        Func<int, int> operation = (n) => printResultsOf(Fibonacci, n);
        //Func<int, int> operation = (n) => printResultsOf(SquareOf, n);

        var f30 = operation(30);
        var f35 = operation(35);
        var f40 = operation(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }
    public static int Fibonacci(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
    public static int SquareOf(int n) => n * n;
}
```

Abstrato

```
using System;
using static System.Console;
class Program
{
    static void Main()
    {
        Func<int, int> printResultsOf(Func<int, int> func) => (n) =>
        {
            var result = func(n);
            WriteLine($"Result for {n} is {result}");
            return result;
        };

        var operation = printResultsOf(Fibonacci);
        //var operation = printResultsOf(SquareOf);

        var f30 = operation(30);
        var f35 = operation(35);
        var f40 = operation(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }
    public static int Fibonacci(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
    public static int SquareOf(int n) => n * n;
}
```

```
using System;
using static System.Console;

namespace Sample
{
    class Program
    {
        static void Main()
        {
            var operation = Decorators.PrintingResultsOf<int, int>(Fibonacci);
            var f30 = operation(30);
            var f35 = operation(35);
            var f40 = operation(40);
            WriteLine($"Results: {f30}, {f35}, {f40}");
        }

        public static int Fibonacci(int n)
        {
            if (n == 0) return 0;
            if (n == 1) return 1;
            return Fibonacci(n - 1) + Fibonacci(n - 2);
        }

        public static int SquareOf(int n) => n * n;
    }
```

Concreto

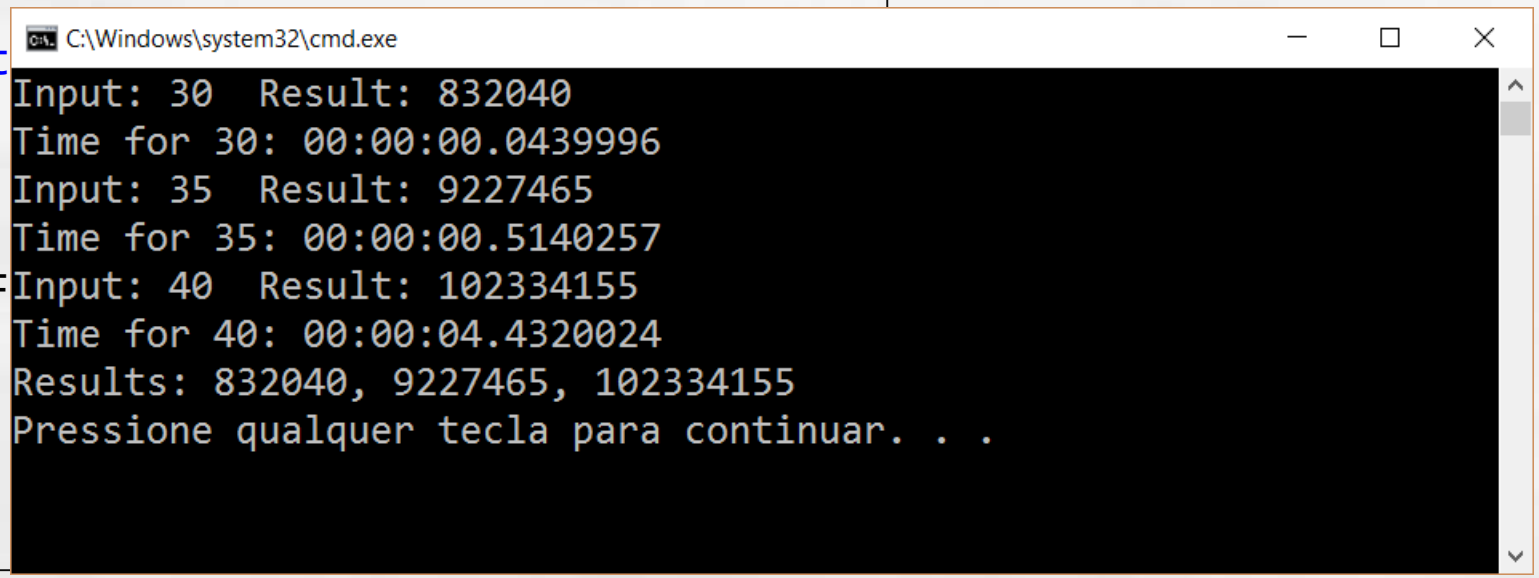
```
    static class Decorators
    {
        public static Func<T, TResult> PrintingResultsOf<T, TResult>(
            this Func<T, TResult> func
        ) => (input) =>
        {
            var result = func(input);
            WriteLine($"Input: {input}  Result: {result}");
            return result;
        };
    }
}
```

Abstrato

```
public static Func<T, TResult> MeasuringTimeOf<T, TResult>(
    this Func<T, TResult> func
) => (input) =>
{
    var before = DateTime.Now;
    var result = func(input);
    var totalTime = DateTime.Now - before;
    WriteLine($"Time for {input}: {totalTime}");
    return result;
};
```

```
class Program
{
    static void Main()
    {
        var operation = ((Func<int, int>)Fibonacci)
            .PrintingResultsOf()
            .MeasuringTimeOf();

        var f30 = operation(30);
        var f35 = operation(35);
        var f40 = operation(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }

    public static int Fibonacci(int n)
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }

    public static int SquareOf(int n) => n * n;
}
```

```
class Program
{
    static void Main()
    {
        var operation = ((Func<int, int>)Fibonacci)
            .PrintingResultsOf()
            .MeasuringTimeOf();

        var f30 = operation(30);
        var f35 = operation(35);
        var f40 = operation(40);
        WriteLine($"Results: {f30}, {f35}, {f40}");
    }

    public static int Fibonacci(int
    {
        if (n == 0) return 0;
        if (n == 1) return 1;
        return Fibonacci(n - 1) + F
    }

    public static int SquareOf(int
}
```



```
using System;
using static System.Console;
class Resource : IDisposable
{
    public Resource() => WriteLine("created ...");
    public void Foo() => WriteLine("Foo");
    public void Fee() => WriteLine("Fee");
    public void Dispose() => WriteLine("cleanup...");
}
```

```
using System;
using static System.Console;
class Resource : IDisposable
{
    public Resource() => WriteLine("created ...");
    public void Foo() => WriteLine("Foo");
    public void Fee() => WriteLine("Fee");
    public void Dispose() => WriteLine("cleanup...");
}
```

```
public class Program
{
    public static void Main()
    {
        using (var r = new Resource())
        {
            r.Foo();
            r.Fee();
        }
    }
}
```

```
using System;
using static System.Console;
class Resource : IDisposable
{
    public Resource() => WriteLine("created ...");
    public void Foo() => WriteLine("Foo");
    public void Fee() => WriteLine("Fee");
    public void Dispose() => WriteLine("cleanup...");
}
```

```
public class Program
{
    public static void Main()
    {
        using (var r = new Resource())
        {
            r.Foo();
            r.Fee();
        }
    }
}
```

```
class Resource
{
    private Resource() { WriteLine("created ..."); }
    public void Foo() { WriteLine("Foo"); }
    public void Fee() { WriteLine("Fee"); }
    private void Dispose() { WriteLine("cleanup..."); }

    public static void Use(Action<Resource> block)
    {
        var r = new Resource();
        try { block(r); } finally { r.Dispose(); }
    }
}
```

```
class Resource
{
    private Resource() { WriteLine("created ..."); }
    public void Foo() { WriteLine("Foo"); }
    public void Fee() { WriteLine("Fee"); }
    private void Dispose() { WriteLine("cleanup..."); }

    public static void Use(Action<Resource> block)
    {
        var r = new Resource();
        try { block(r); } finally { r.Dispose(); }
    }
}
```

```
public class Program
{
    public static void Main()
    {
        Resource.Use(r => {
            r.Foo();
            r.Fee();
        });
    }
}
```

LAZYNESS

```
public static string[] GetNamesFromEmployees(Employee[] employees)
{
    var result = new string[employees.Length];
    for (var i = 0; i < employees.Length; i++)
    {
        result[i] = employees[i].Name;
    }
    return result;
}
```

```
public static string[] GetNamesFromEmployees(Employee[] employees)
{
    var result = new string[employees.Length];
    for (var i = 0; i < employees.Length; i++)
    {
        result[i] = employees[i].Name;
    }
    return result;
}
```

```
public static List<string> GetNamesFromEmployees(List<Employee> employees)
{
    var result = new List<string>(employees.Count);
    foreach (var employee in employees)
    {
        result.Add(employee.Name);
    }
    return result;
}
```

```
public static List<string> GetNamesFromEmployees(List<Employee> employees)
{
    var result = new List<string>(employees.Count);
    foreach (var employee in employees)
    {
        result.Add(employee.Name);
    }
    return result;
}
```

```
public static IList<string> GetNamesFromEmployees(IList<Employee> employees)
{
    var result = new List<string>(employees.Count);
    foreach (var employee in employees)
    {
        result.Add(employee.Name);
    }
    return result;
}
```

```
public static IList<string> GetNamesFromEmployees(IList<Employee> employees)
{
    var result = new List<string>(employees.Count);
    foreach (var employee in employees)
    {
        result.Add(employee.Name);
    }
    return result;
}
```

```
public static IEnumerable<string> GetNamesFromEmployees(IEnumerable<Employee> employees)
{
    var result = new List<string>();
    foreach (var employee in employees)
    {
        result.Add(employee.Name);
    }
    return result;
}
```

```
public static IEnumerable<string> GetNamesFromEmployees(IEnumerable<Employee> employees)
{
    var result = new List<string>();
    foreach (var employee in employees)
    {
        result.Add(employee.Name);
    }
    return result;
}
```

```
public static IEnumerable<string> GetNamesFromEmployees(IEnumerable<Employee> employees)
{
    var result = new List<string>();
    foreach (var employee in employees)
    {
        result.Add(employee.Name);
    }
    return result;
}
```

```
public static IEnumerable<string> GetNamesFromEmployees(IEnumerable<Employee> employees)
{
    var result = new List<string>();
    foreach (var employee in employees)
    {
        result.Add(employee.Name);
    }
    return result;
}
```

```
public static IEnumerable<string> GetNamesFromEmployees(IEnumerable<Employee> employees)
{
    foreach (var employee in employees)
    {
        yield return employee.Name;
    }
}
```

Abstrato

PAUSA PARA ENTENDER...

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
1
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
1
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
1
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
1
After Yield 1
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
1
After Yield 1
Before Yield 2
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
Before calling Get4
After calling Get4
Before Yield 1
1
After Yield 1
Before Yield 2


...
```

VOLTAMOS...

```
public static IEnumerable<string> GetNamesFromEmployees(IEnumerable<Employee> employees)
{
    foreach (var employee in employees)
    {
        yield return employee.Name;
    }
}
```

```
public static IEnumerable<string> GetNamesFromEmployees(IEnumerable<Employee> employees)
{
    foreach (var employee in employees)
    {
        yield return employee.Name;
    }
}
```

```
public static class EnumerableOfEmployees
{
    public static IEnumerable<string> GetNames(IEnumerable<Employee> employees)
    {
        foreach (var employee in employees)
        {
            yield return employee.Name;
        }
    }
}
```

```
EnumerableOfEmployees.GetNames(someListOfEmployees);
```

```
public static class EnumerableOfEmployees
{
    public static IEnumerable<string> GetNames(
        this IEnumerable<Employee> employees
        )
    {
        foreach (var employee in employees)
        {
            yield return employee.Name;
        }
    }
}
```

```
var names = EnumerableOfEmployees.GetNames(someListOfEmployees);
```

```
var names = someListOfEmployees.GetNames();
```

```
public static class EnumerableOfEmployees
{
    public static IEnumerable<string> GetNames(
        this IEnumerable<Employee> employees
        )
    {
        foreach (var employee in employees)
        {
            yield return employee.Name;
        }
    }

    public static IEnumerable<string> GetSocialSecurityNumbers(
       this IEnumerable<Employee> employees
       )
    {
        foreach (var employee in employees)
        {
            yield return employee.SocialSecurityNumber;
        }
    }
}
```

```
public static class EnumerableOfEmployees
{
    public static IEnumerable<string> GetNames(
        this IEnumerable<Employee> employees
        )
    {
        foreach (var employee in employees)
        {
            yield return employee.Name;
        }
    }

    public static IEnumerable<string> GetSocialSecurityNumbers(
       this IEnumerable<Employee> employees
       )
    {
        foreach (var employee in employees)
        {
            yield return employee.SocialSecurityNumber;
        }
    }
}
```

```
public static class EnumerableOfEmployees
{
    public static IEnumerable<string> GetNames(
        this IEnumerable<Employee> employees
        )
    {
        foreach (var employee in employees)
        {
            yield return employee.Name;
        }
    }

    public static IEnumerable<string> GetSocialSecurityNumbers(
       this IEnumerable<Employee> employees
       )
    {
        foreach (var employee in employees)
        {
            yield return employee.SocialSecurityNumber;
        }
    }
}
```

Abstrato

```
public static class EnumerableOfEmployees
{
    public static IEnumerable<TResult> Get<TResult>(
        this IEnumerable<Employee> employees,
        Func<Employee, TResult> selector
        )
    {
        foreach (var employee in employees)
        {
            yield return selector(employee);
        }
    }
}
```

```
var names = someListOfEmployees.Get(e => e.Name);
var ssn = someListOfEmployees.Get(e => e.SocialSecurityNumber);
```

```
public static class EnumerableOfEmployees
{
    public static IEnumerable<TResult> Get<TResult>(
        this IEnumerable<Employee> employees,
        Func<Employee, TResult> selector
        )
    {
        foreach (var employee in employees)
        {
            yield return selector(employee);
        }
    }
}
```

```
public static class EnumerableOfEmployees
{
    public static IEnumerable<TResult> Get<TResult>(
        this IEnumerable<Employee> employees,
        Func<Employee, TResult> selector
        )
    {
        foreach (var employee in employees)
        {
            yield return selector(employee);
        }
    }
}
```

```
public static class Enumerable
{
    public static IEnumerable<TResult> Get<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

Abstrato

```
public static class Enumerable
{
    public static IEnumerable<TResult> Get<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

Abstrato

```
public static class Enumerable
{
    public static IEnumerable<TResult> Get<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

```
public static class Enumerable
{
    public static IEnumerable<TResult> Select<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

```
public static class Enumerable
{
    public static IEnumerable<TResult> Select<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

LINQ

Para objetos em memória

```
public static class Enumerable
{
    public static IEnumerable<TResult> Select<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

```
var names = someListOfEmployees.Select(e => e.Name);
var ssn = someListOfEmployees.Select(e => e.SocialSecurityNumber);
```

```
public static IEnumerable<T> Where<T>(
    this IEnumerable<T> elements,
    Func<T, bool> filter
)
{
    foreach (var element in elements)
    {
        if (filter(element))
        {
            yield return element;
        }
    }
}
```

```
public static IEnumerable<T> Take<T>(
    this IEnumerable<T> elements,
    int count
)
{
    var i = 0;
    foreach (var element in elements)
    {
        if (i >= count) yield break;
        yield return element;
        i++;
    }
}
```

```
public static class Enumerable
{
    public static IEnumerable<TResult> Select<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

```
public interface IEnumerable<out T> : IEnumerable
{
    IEnumerator<T> GetEnumerator();
}
```

```
public static class Enumerable
{
    public static IEnumerable<TResult> Select<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

```
public interface IEnumerator<out T> : IDisposable, IEnumerator
{
    T Current { get; }
}
```

```
public interface IEnumerable<out T> : IEnumerable
{
    IEnumerator<T> GetEnumerator();
}
```

```
public static class Enumerable
{
    public static IEnumerable<TResult> Select<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

```
public interface IEnumerator
{
    object Current { get; }
    bool MoveNext();
    void Reset();
}
```

```
public interface IEnumerator<out T> : IDisposable, IEnumerator
{
    T Current { get; }
}
```

```
public interface IEnumerable<out T> : IEnumerable
{
    IEnumerator<T> GetEnumerator();
}
```

```
public static class Enumerable
{
    public static IEnumerable<TResult> Select<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
```

```
public static class Enumerable
{
    public static IEnumerable<TResult> Select<T, TResult>(
        this IEnumerable<T> elements,
        Func<T, TResult> selector
        )
    {
        foreach (var element in elements)
        {
            yield return selector(element);
        }
    }
}
```

```
public static IEnumerable<TResult> Select<T, TResult>(
    this IEnumerable<T> elements,
    Func<T, TResult> selector
    )
{
    using (var enumerator = elements.GetEnumerator())
    {
        while (enumerator.MoveNext())
        {
            yield return selector(enumerator.Current);
        }
    }
}
```

PAUSA PARA ENTENDER...

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
public static IEnumerable<int> Get4()
        => new Get4Enumerable();
```

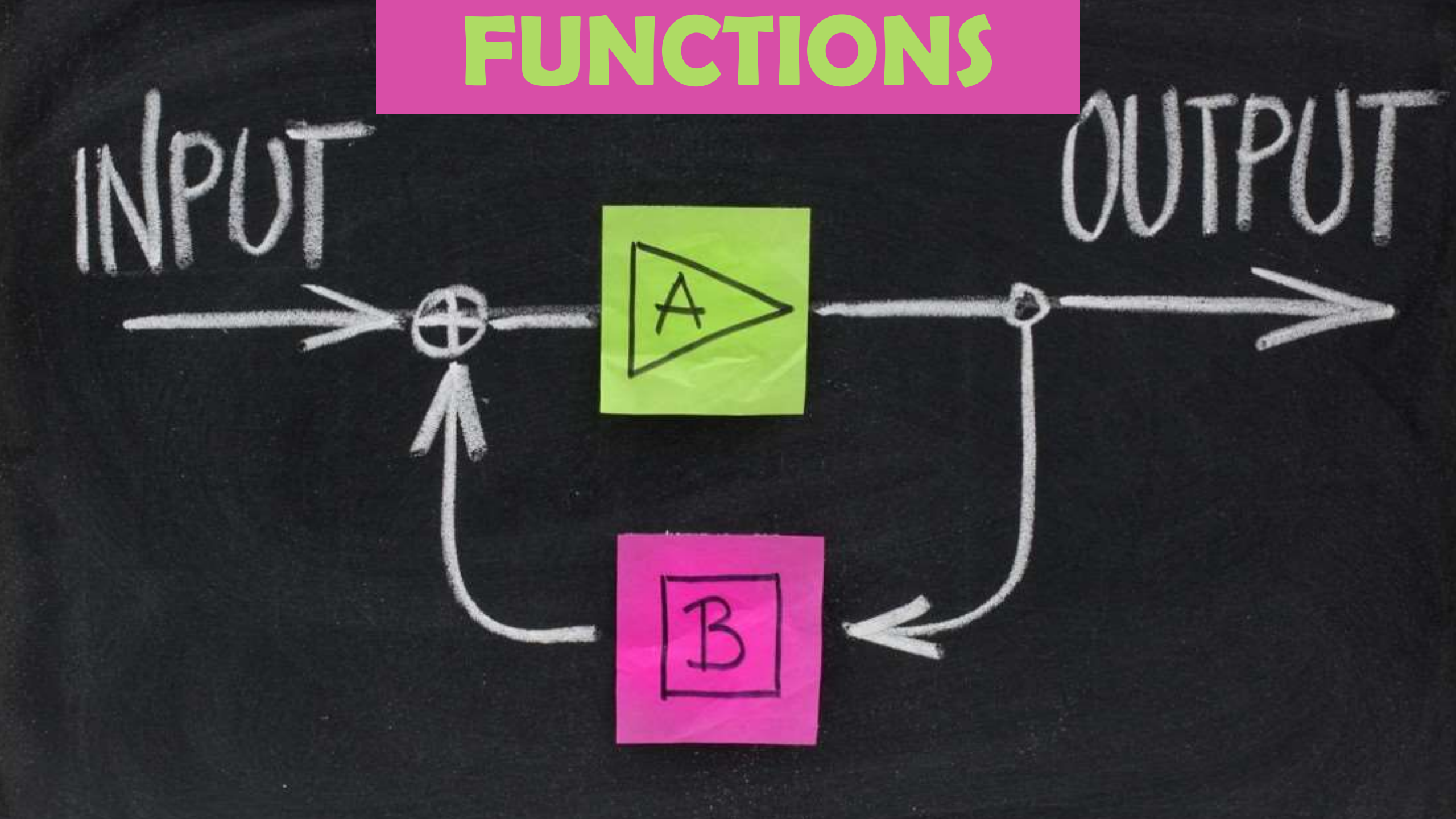
```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
using (var enumerator = elements.GetEnumerator())
{
    while (enumerator.MoveNext())
    {
        WriteLine(enumerator.Current);
    }
}
```

```
using System.Collections.Generic;
using static System.Console;

public class Program
{
    public static void Main()
    {
        WriteLine("Before calling Get4");
        var elements = Get4();
        WriteLine("After calling Get4");
        foreach (var e in elements)
        {
            WriteLine(e);
        }
    }

    public static IEnumerable<int> Get4()
    {
        for (var i = 1; i <= 4; i++)
        {
            WriteLine($"Before Yield {i}");
            yield return i;
            WriteLine($"After Yield {i}");
        }
    }
}
```

```
class Get4Enumerator : IEnumerator<int>
{
    private int _state;
    private int _current;
    public void Dispose() { }
    bool IEnumerator.MoveNext()
    {
        if (_state == 1) WriteLine($"After Yield {_current}");

        if (_current >= 4)
        {
            _state = 2;
            return false;
        }

        _current++;
        WriteLine($"Before Yield {_current}");
        _state = 1;
        return true;
    }

    public void Reset() { _current = 0; }
    public int Current => _current;
    object IEnumerator.Current => Current;
}
```

FUNCTIONS



```
public interface IArraySortStrategy
{
    T[] Sort<T>(T[] input, Comparison<T> comparison);
}
```

```
public class QuickSortStrategy : IArraySortStrategy
{
    public T[] Sort<T>(T[] input, Comparison<T> comparison)
    {  /* .. */ }
}
public class MergeSortStrategy : IArraySortStrategy
{
    public T[] Sort<T>(T[] input, Comparison<T> comparison)
    {  /* .. */ }
}
```

```
namespace HelloStrategy.Controllers
{
    [Route("api/[controller]")]
    public class ValuesController : Controller
    {
        private readonly IArraySortStrategy _sortingStrategy;

        public ValuesController(IArraySortStrategy sortingStrategy)
        {
            _sortingStrategy = sortingStrategy;
        }
        // ..
    }
}
```

```
public delegate T[] SortingAlgorithm<T>(
    T[] input,
    Comparison<T> comparison
);
```

```
public static class SortingImplementations
{
    public static T[] QuickSort<T>(
        T[] input,
        Comparison<T> comparison
    )
    {  /* .. */ }

    public static T[] MergeSort<T>(
        T[] input,
        Comparison<T> comparison
    )
    {  /* .. */ }
}
```

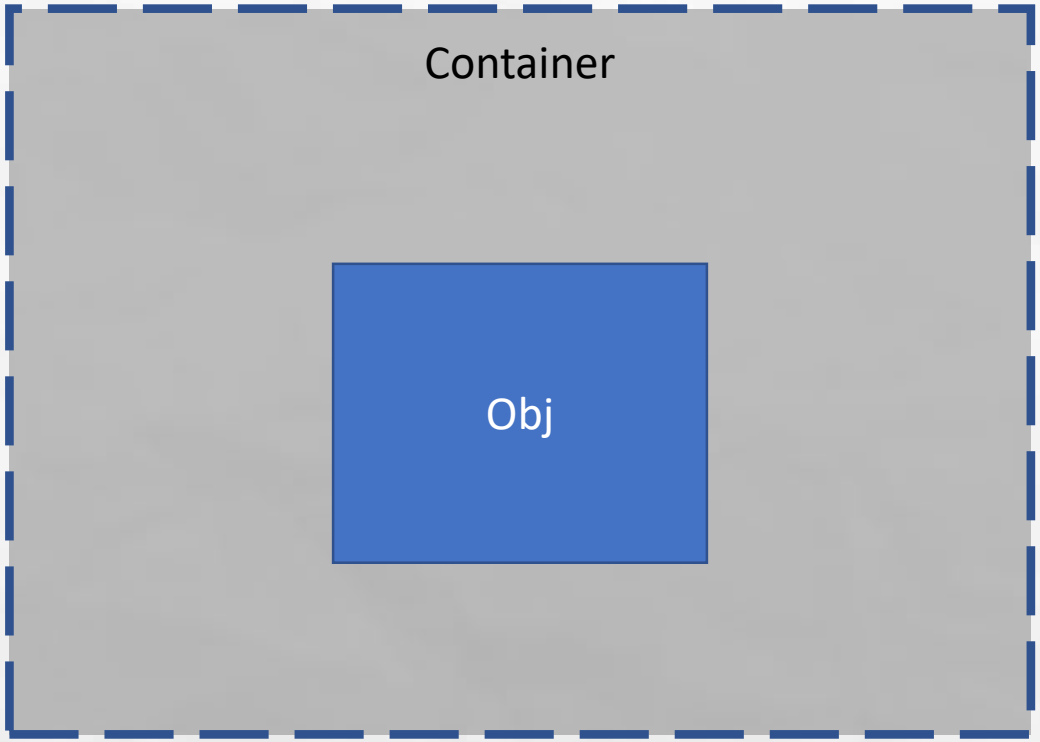
```
namespace HelloStrategy.Controllers
{
    [Route("api/[controller]")]
    public class ValuesController : Controller
    {
        private readonly SortingAlgorithm<Customer> _sa;

        public ValuesController(SortingAlgorithm<Customer> sa)
        {
            _sa = sa;
        }

        // ..
    }
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<SortingAlgorithm<Customer>>(
        SortingImplementations.QuickSort
    );

    // Add framework services.
    services.AddMvc();
}
```


CONTAINERS


NÃO

Abstrato


Container
Obj

Task<T>

IEnumerable<T>

```
public interface IEmployeeRepository
{
    Employee GetById(string id);
}


class EmployeeRepository : IEmployeeRepository
{
    public Employee GetById(string id)
        => new DbContext().Find(id);
}
```

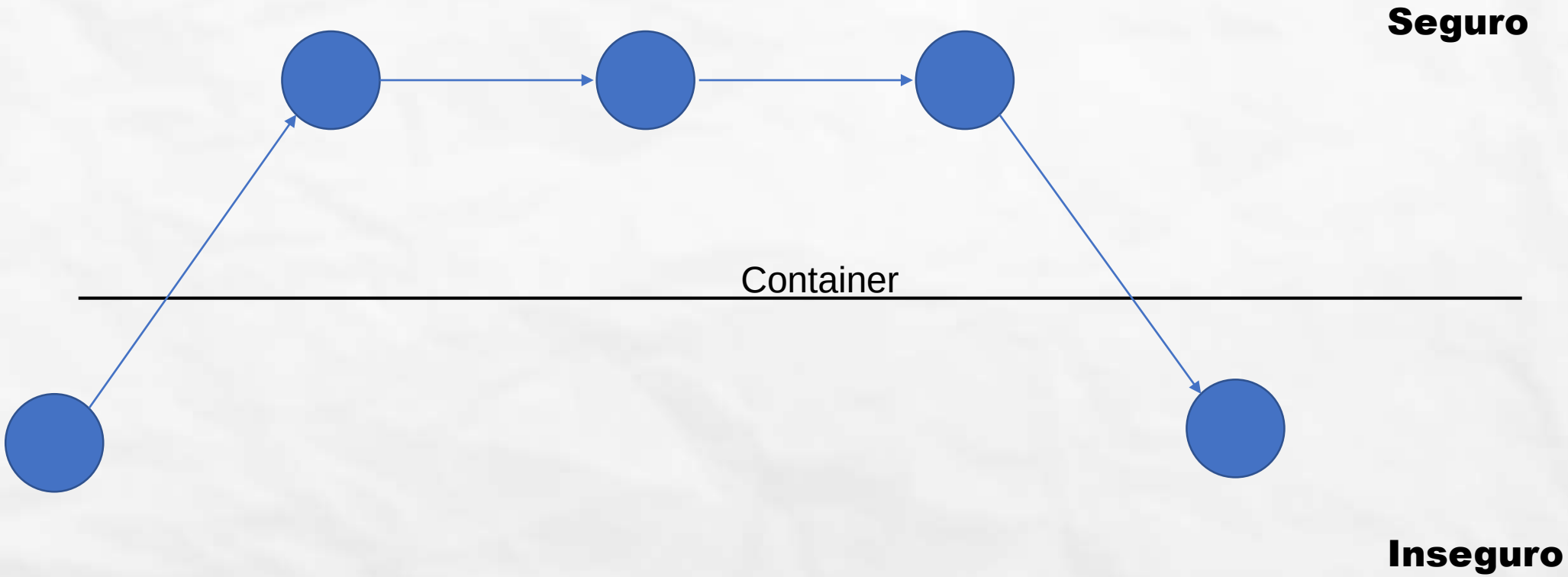
```
public IActionResult Get(string id)
{
    var employee = _repository.GetById(id);

    if (employee == null)
    {
        return NotFound();
    }

    return Ok(employee);
}
```

```
public struct Option<T>
{
    internal T Value { get; }
    public bool IsSome { get; }
    public bool IsNone => !IsSome;

    internal Option(T value, bool isSome)
    {
        Value = value;
        IsSome = isSome;
    }

    public TR Match<TR>(Func<T, TR> some, Func<TR> none)
        => IsSome ? some(Value) : none();

    public void Match(Action<T> some, Action none)
    {
        if (IsSome) some(Value); else none();
    }
}
```

```
public static class Option
{
    public static Option<T> Of<T>(T value)
        => new Option<T>(value, value != null);
}
```

```
public interface IEmployeeRepository
{
    Option<Employee> GetById(string id);
}

class EmployeeRepository : IEmployeeRepository
{
    public Option<Employee> GetById(string id) =>
        Option.Of(new DbContext().Find(id));
}
```

```
public IActionResult Get(string id) =>
    _repository.GetById(id).Match(
        some: e => Ok(e),
        none: () => NotFound()
    );
```


Seguro
Container
Inseguro


Abstrato
Concreto


ASK MORE
QUESTIONS

OBRIGADO!

ELEMAR JR

falecom@elemarjr.com
elemarjr@ravendb.net
elemarjr.com