
We’ve got this. Together.

# Functional Programming In C#

Johan Olsson

# C# is an OOP language

Gets more FP features with each new version.

Tuples, Pattern matching, Local functions, Switch expressions, Records, ...

# Let’s go on an adventure and explore

Hopefully, we find something interesting to bring home.

# Johan Olsson

# 30+ years in the business

# Discovered FP about 8 years ago.

# What is FP, anyway?

# Different Paradigm

## Functional vs Object-oriented

# OOP

Combine Data and Behavior into objects
Use inheritance for variations
Imperative style

# FP

Separate Data and Behavior
Use composition for variations
Declarative style

# Immutable Data

# Immutable Data

```
class Rectangle
{
    public Rectangle(int width, int height)
    {
        Width = width;
        Height = height;
    }

    public int Width { get; }
    public int Height { get; }
}
```

# Immutable Data

```
record Rectangle(
    int Width,
    int Height);
```

- Value based equality
- Create copy using ’with’ keyword

```
var a = new Rectangle(1, 2);
var b = a with { Height = 5 };
```

# Immutable Data

Why?

- Easy to reason about.
  Can pass data to functions without side effects.

- Thread Safe.
  Can share data between threads without locks.

# Functions are first class citizens in C#

# Higher Order Function

A function that takes another function as a parameter or returns a function (or both).

# HOF by Example

```
var result = new List<Rectangle>();
foreach (var r in rectangles)
{
    if (r.Height > 5 || r.Width > 5)
    {
        result.Add(r);
    }
}
return result;
```

# HOF by Example

LINQ:

```
return rectangles.Where(r => r.Height > 5 || r.Width > 5);
```

# HOF by Example

```
return rectangles.Where(SideIsGreaterThan(5));
```

```
Func<Rectangle, bool> SideIsGreaterThan(int limit)
{
    return r => r.Height > limit || r.Width > limit;
}
```

# HOF by Example

```
return rectangles.Where(SideIsGreaterThan(5));
```

```
Func<Rectangle, bool> SideIsGreaterThan(int limit) =>
   r => r.Height > limit || r.Width > limit;
```

# Pipelining

a.k.a. call chaining

# Pipelining

OS shells, (like PowerShell), have a pipeline operator

```
PS> type myfile.txt | sort | select –First 3
```

# Call chaining

C# does not have any pipeline operator. We have to do call chaining instead.

Common in ASPNET

```
services.AddAuthentication().AddCaching().AddControllers();
```

# Call chaining

Implemented using extension methods.

```
public static IServiceCollection AddAuthentication(
   this IServiceCollection services)
{
   services.DoSomething();
   return services;
}
```

# Call chaining

Makes code more readable

```
customer
    .GetLatestOrderFrom(database)
    .GiveDiscount(10)
    .SaveChangesTo(database);
```

# Call chaining

Easy to compose new logic

```
customer
    .GetLatestOrderFrom(database)
    .GiveDiscount(5)
    .MoveDeliveryDateTo(newDate)
    .SaveChangesTo(database);
```

# Expressions

# Switch Expression

# Switch Statement

```
switch (something)
{
    case 1:
        result = DoA();
        break;
    case 2:
        result = DoB();
        break;
}
```

# Switch Expression

```
result = something switch
{
    1 => DoA(),
    2 => DoB(),
};
```

# Switch Expression

Cool, another way to write switches.

**But,** that’s not what you use it for.

Use it to simplify complex logic.

# Complex Logic

```
if (something)
{
    result = DoA();
}
else if (anotherThing)
{
    result = DoB();
}
else
{
    result = DoC();
}
```

# Complex Logic

```
result = (something, anotherThing) switch
{
    (true, _) => DoA(),
    (false, true) => DoB(),
    (false, false) => DoC(),
};
```

# Complex Logic

```
result = customer switch
{
    { Name: "Batman“ } => DoA(),
    { OrderTotal: > 5 } => DoB(),
    _ => DoC()
};
```

# Switch Expression and pattern matching

Code is easier to read.

Compiler can warn if you missed a case.

# Fluent Api

# Fluent Api

Build a fluent Api by combining:

- Higher Order Functions
- Call chaning
- Expressions

# Fluent Api, Example

FluentValidation
- NuGet package
- Build strongly-typed validation rules

# Fluent Api, Example

```
RuleFor(input => input.Name)
    .NotEmpty()
    .WithMessage("Please enter name");
```

# Fluent Api, Example

```
public class MyValidator : AbstractValidator<Customer>
{
    public MyValidator()
    {
        RuleFor(input => input.Name)…
        RuleFor(input => input.Adress)…
   }
}

var result = new MyValidator().Validate(input);
if (!result.IsValid)
    return result.Errors;
```

# Fluent Api, Example

```
RuleFor(t => t.Weight)
    .NotNull()
    .WithMessage(localizer[RequiredIfTimeOn])
    .When(t => t.TimeOn.HasValue)
    .Unless(t => t.WeightNotApplicable);
```

# Fluent Api

- Less code
- Easier to read

# What have we found?

# Functional features and style makes code easier to read.

# Easy to read = Easy to understand

# Easy to understand = Less bugs

# Functional Programming => Less Bugs

# Questions?

## Questions & Discussion







# Thank you!