En skonsam inkörsport till funktionell programmering i C#

2023-01-12
Peter Dahlgren

Objektorienterad programmering

(imperativ)

Funktionell programmering

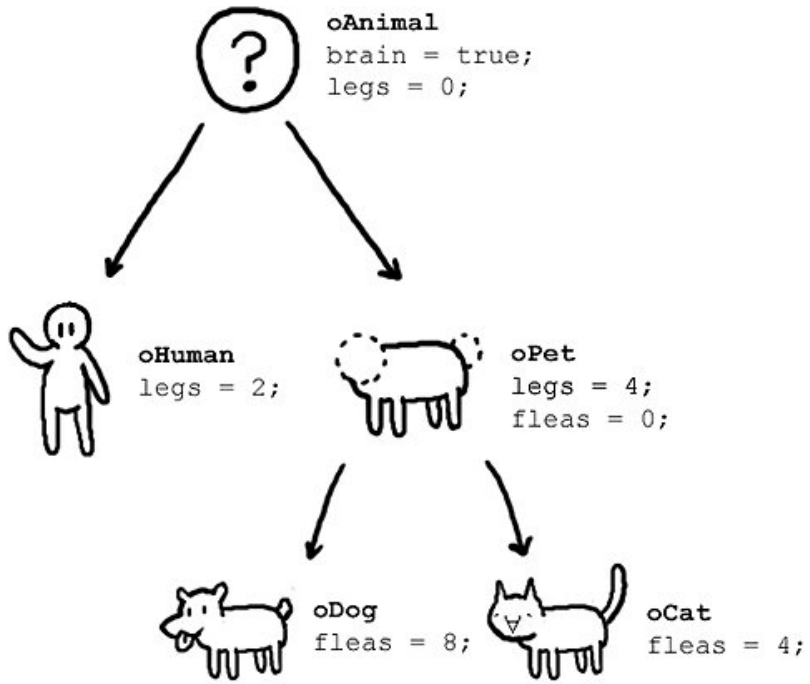
Objektorienterad programmering

(imperativ)

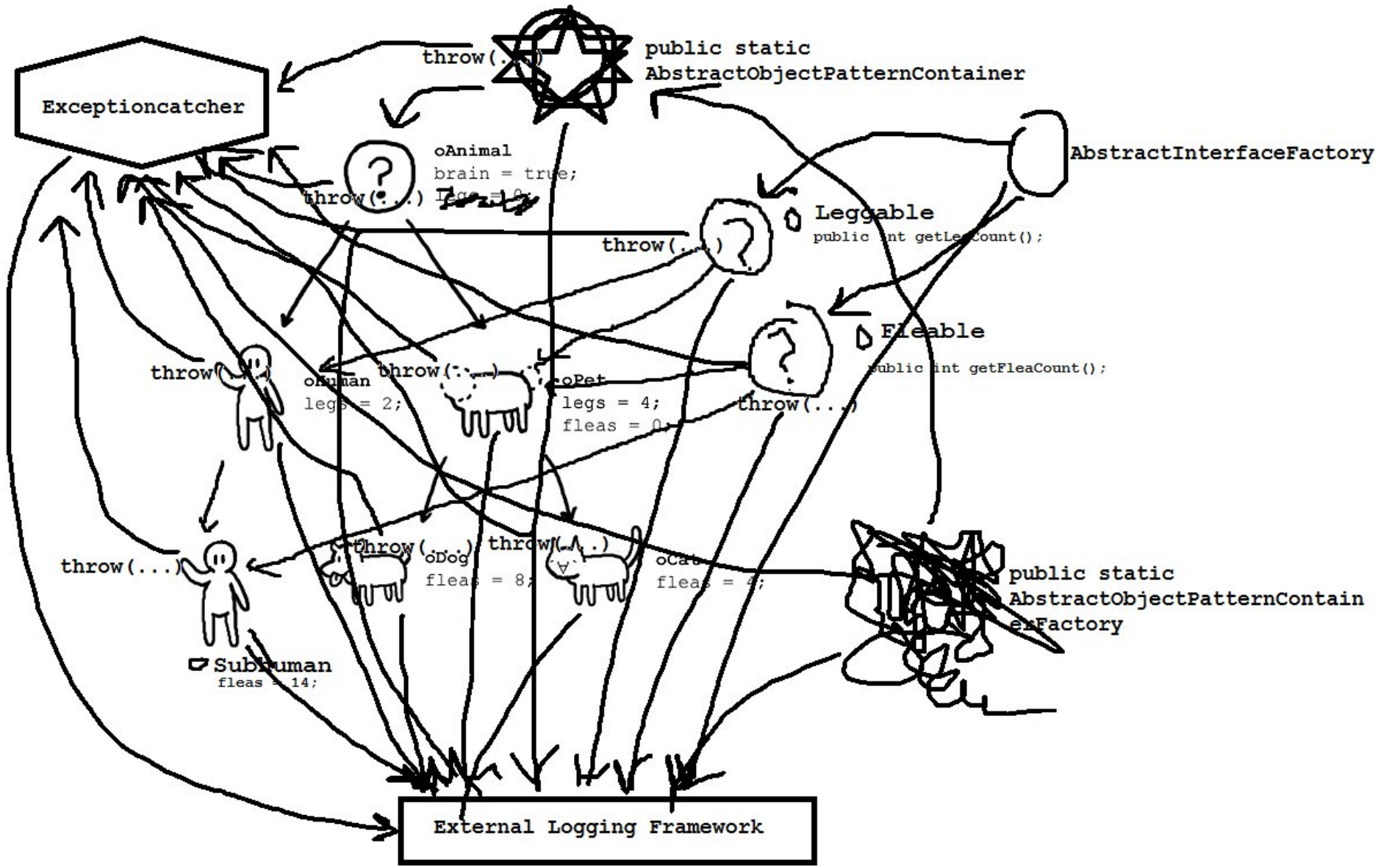
1. Skillnaden mellan **OOP** och **FP**
2. Ni använder redan lite **FP**
3. Lösa problem med **FP**

Funktionell programmering

What OOP users claim



What actually happens



OOP	FP
Objekt	Funktioner
Imperativ: ”gör så här, steg för steg”	Deklarativ: ”ge mig det här, jag bryr mig inte hur”
State	Stateless
Mutable	Immutable
if & switch	Funktioner, pattern matching
Abstraktion => polymorfism	Abstraktion => komposition
Spaghettikod	Mer funktioner

Varför FP ?

- Pure
- Immutable
- Deterministic
- Honest functions
- Higher-order functions

Varför FP ?

- Pure
- Immutable
- Deterministic
- Honest functions
- Higher-order functions

- Inga states
- Högre abstraktion
- Kodåteranvändning
- Inga bieffekter (*side effects*)
- Samma beteende varje gång
- Om en funktion körs nu eller senare spelar ingen roll
- Lättare att komponera ihop flera funktioner
- Enklare att resonera kring funktioner
- Lättare att parallellisera
- Enklare att testa

Varför FP ?

- Pure
- Immutable
- Deterministic
- Honest functions
- Higher-order functions

- Kalla på många funktioner → större stack
- Högre abstraktion → svårare att debugga
- Abstraktion → något långsammare
- I/O-operationer → alltid bieffekter

Exempelvis databaser, filer & nätverk

OOP

```
public int GetDay(int addDays)
{
    _days = DateTime.Now.AddDays(addDays).Day;
    return _days;
}
```

OOP

Ändrar global state
mutable, bieffekt, oärlig

```
public int GetDay(int addDays)
{
    _days = DateTime.Now.AddDays(addDays).Day;
    return _days;
}
```

Hämtar datum utanför funktionen
oren, icke-deterministisk

FP

Allt som behövs kommer som indata

```
public int GetDay(int addDays, DateTime dt)
{
    return dt.AddDays(addDays).Day;
}
```

Returnerar nytt värde
ändrar inget

Allt görs inom funktionen
ren, deterministisk

FP

Gör något

Statements

Exempel:

- `Console.WriteLine()`
- `File.Write()`
- `if`
- `for/foreach`

Returnerar något

Expressions

Exempel:

- `string.ToLower()`
- `.Select(x => x)`
- `bool ? true : false`
- `return x switch`
  (pattern matching)

FP i C#

Delegates

Action, Func, Predicate

Lambda expressions

Extension methods

Pattern matching C# 7–8

Records C# 9

LINQ

Task<T>, await, Lazy<T>

OOP

```
var items = new List<string>()
{
    "a", "b", "c"
};

foreach (var x in items)
{
    if (x == "a")
    {
        x = x.ToUpper();
        Console.WriteLine(x);
    }
};
```

FP

```
var items = new List<string>()
{
    "a", "b", "c"
};

items
    .Where(x => x == "a")
    .Select(x => x.ToUpper());
    .ToList()
    .ForEach(Console.WriteLine);
```

OOP

```
var items = new List<string>()
{
    "a", "b", "c"
};

foreach (var x in items)
{
    if (x == "a")
    {
        x = x.ToUpper();
        Console.WriteLine(x);
    }
};
```

Steg-för-steg instruktioner

FP

```
var items = new List<string>()
{
    "a", "b", "c"
};

items
    .Where(x => x == "a")
    .Select(x => x.ToUpper());
    .ToList()
    .ForEach(Console.WriteLine);
```

OOP

```
var items = new List<string>()
{
    "a", "b", "c"
};

foreach (var x in items)
{
    if (x == "a")
    {
        x = x.ToUpper();
        Console.WriteLine(x);
    }
};
```

Ändrar state (mutable)

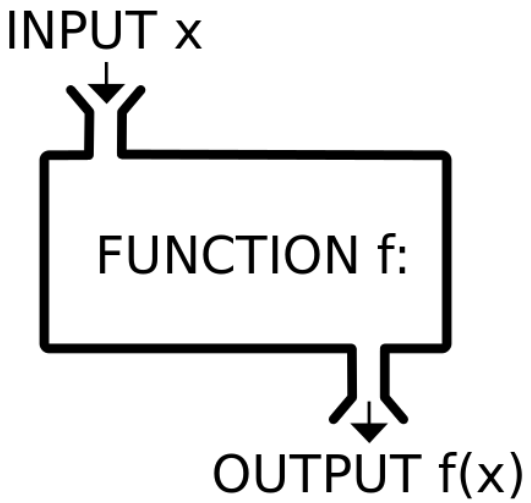
FP

```
var items = new List<string>()
{
    "a", "b", "c"
};

items
    .Where(x => x == "a")
    .Select(x => x.ToUpper());
    .ToList()
    .ForEach(Console.WriteLine);
```

OOP

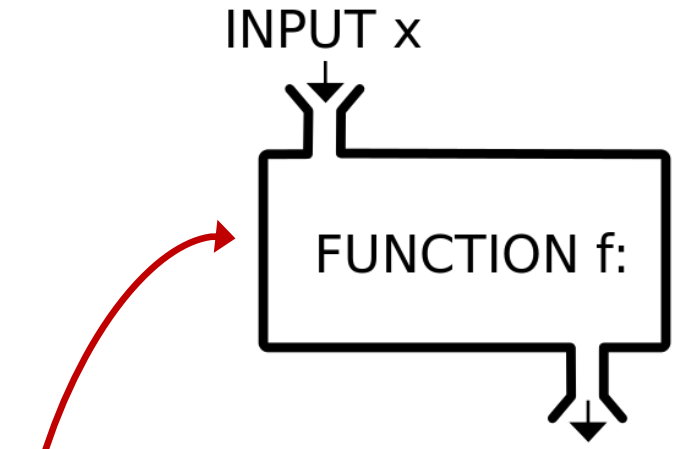
25



30

FP

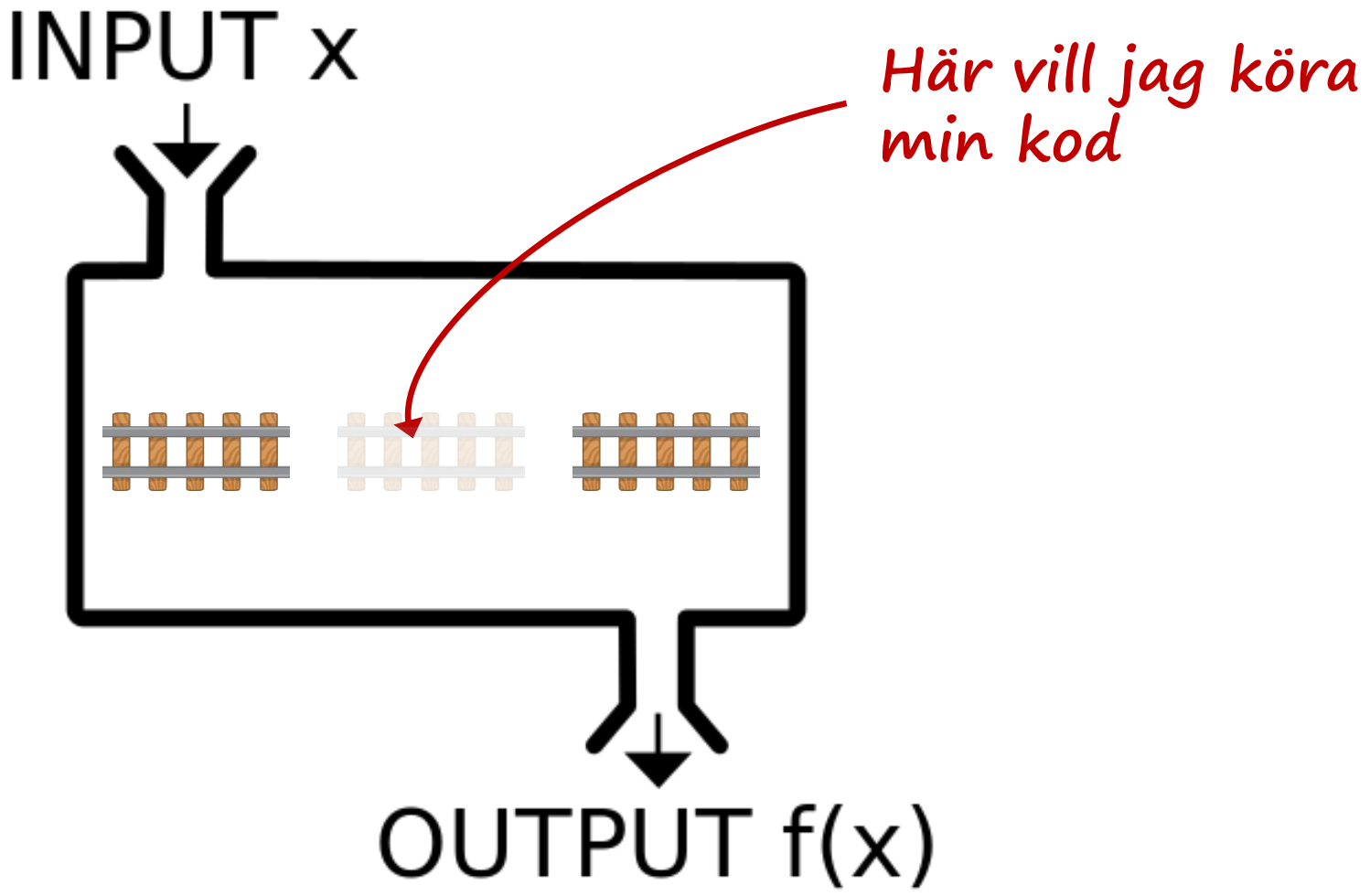
funktion funktion



funktion

Higher-order function (HOF)

Kör i mitten-pattern



FP

```
static void Timer()
{
    var sw = new Stopwatch();
    sw.Start();
    workload();
    sw.Stop();
    Console.WriteLine($"Function took {sw.Elapsed} to run");
}

Timer();
```

FP

Action = *void*

```
static void Timer(Action workload)
{
    var sw = new Stopwatch();
    sw.Start();
    workload();
    sw.Stop();
    Console.WriteLine($"Function took {sw.Elapsed} to run");
}

Timer(() => MyExpensiveFunction());
```

FP

Kan ta emot indata också

```
static void Timer(Action workload, Action<Stopwatch> write)
{
    var sw = new Stopwatch();
    sw.Start();
    workload();
    sw.Stop();
    write(sw);
}

Timer(() => MyExpensiveFunction(),
      (sw) => Console.WriteLine($"Function took {sw.Elapsed} to run"));
```

FP

Func = *return*

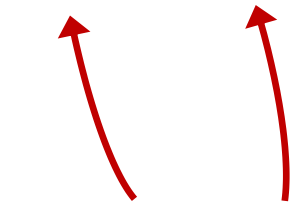
```
static string Timer(Action workload, Func<Stopwatch, string> output)
{
    var sw = new Stopwatch();
    sw.Start();
    workload();
    sw.Stop();
    return output(sw);
}

string result = Timer(() => MyExpensiveFunction(),
                      (sw) => $"Function took {sw.Elapsed} to run");
```

FP

Lär dig Action & Func

Exempel	Metodsignatur
`Action`	`void metod()`
`Action<string>`	`void metod(string)`
`Action<string, int>`	`void metod(string, int)`
`Func<int>`	`int metod()`
`Func<string, int>`	`int metod(string)`
`Func<bool, string, int>`	`int metod(bool, string)`



Sista typen i Func
är returtypen

(`Predicate` är bara en glorifierad `Func<T, bool>` så strunta i `Predicate` också)

FP


```
Func<int, bool> isNegative = (int x) => x < 0;

Console.WriteLine(isNegative(-5));
```

FP


```
Func<int, bool> isNegative = (int x) => x < 0;

Console.WriteLine(isNegative(-5));
```

FP

```
// Kan återanvändas:
Func<User, bool> isAdmin = (User x) => ...;
Func<User, bool> isAuthenticated = (User x) => ...;
Func<User, bool> isAuthorized = (User x) => ...;


// Lättare att läsa:
users.Where(x => isAdmin(x),
            x => isAuthenticated(x),
            x => isAuthorized(x));
```

OOP

```
public static string Calculate(int value)
{
    decimal result = 0.0m;
    string message = "";

    try
    {
        result = (decimal)1 / value;
        message = "Success";
    }
    catch (Exception ex)
    {
        message = ex.Message;
    }

    return $"Result: {result}, message: {message}";
}
```

Exempel från Darren Hale https://www.youtube.com/watch?v=kPRX_N37DOw

FP

```
public static IResult TryCatch(Func<decimal> workload)
{
    try
    {
        var result = new SuccessResult();
        result.Value = workload();
        result.Message = "Success";
        return result;
    }
    catch (Exception ex)
    {
        var result = new FailedResult();
        result.Value = 0.0m;
        result.Message = ex.Message;
        return result;
    }
}

var result = TryCatch(() => 1 / value);
```

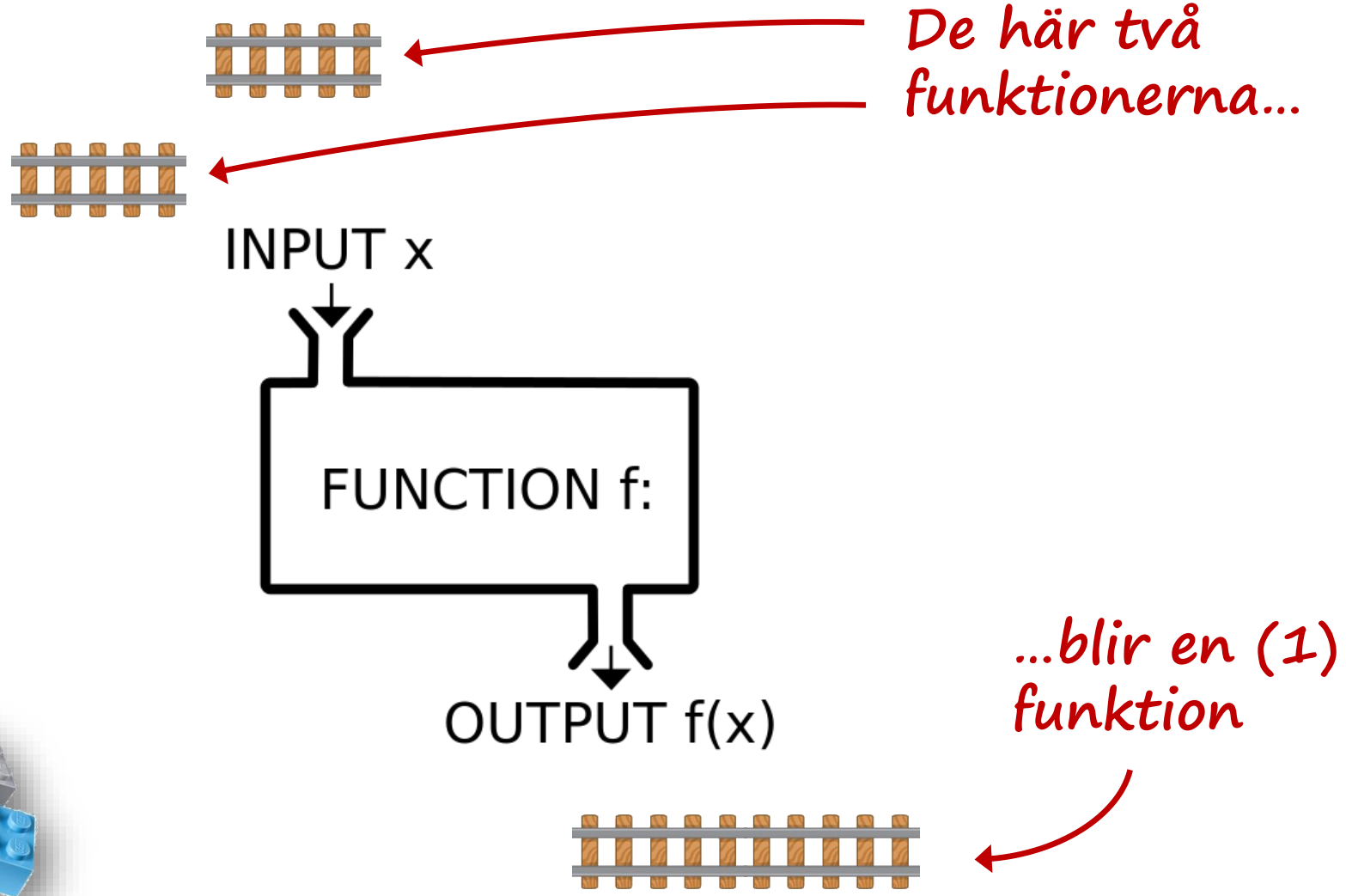
Exempel från Darren Hale https://www.youtube.com/watch?v=kPRX_N37DOw

FP

```
public static IResult<T> TryCatch<T>(Func<T> workload)
{
    try
    {
        var result = new SuccessResult<T>();
        result.Value = workload();
        result.Message = "Success";
        return result;
    }
    catch (Exception ex)
    {
        var result = new FailedResult<T>();
        result.Value = default(T);
        result.Message = ex.Message;
        return result;
    }
}

var result = TryCatch<decimal>(() => 1 / value);
```

Komposition (slå ihop funktioner)

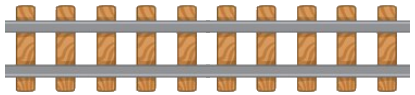


OOP

```
public int Advent(int i) => i + 1;

public int Combined(int a)
{
    return Advent(Advent(a));
}

var andraAdvent = Combined(0);
```



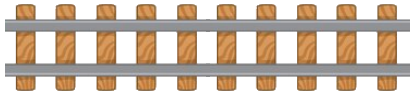
FP

```
Func<T, T> Compose<T>(params Func<T, T>[] xs)
                    => xs.Aggregate((accum, item) => x => accum(item(x)));
```

```
var Combined = Compose(Advent, Advent);

var andraAdvent = Combined(0);
```

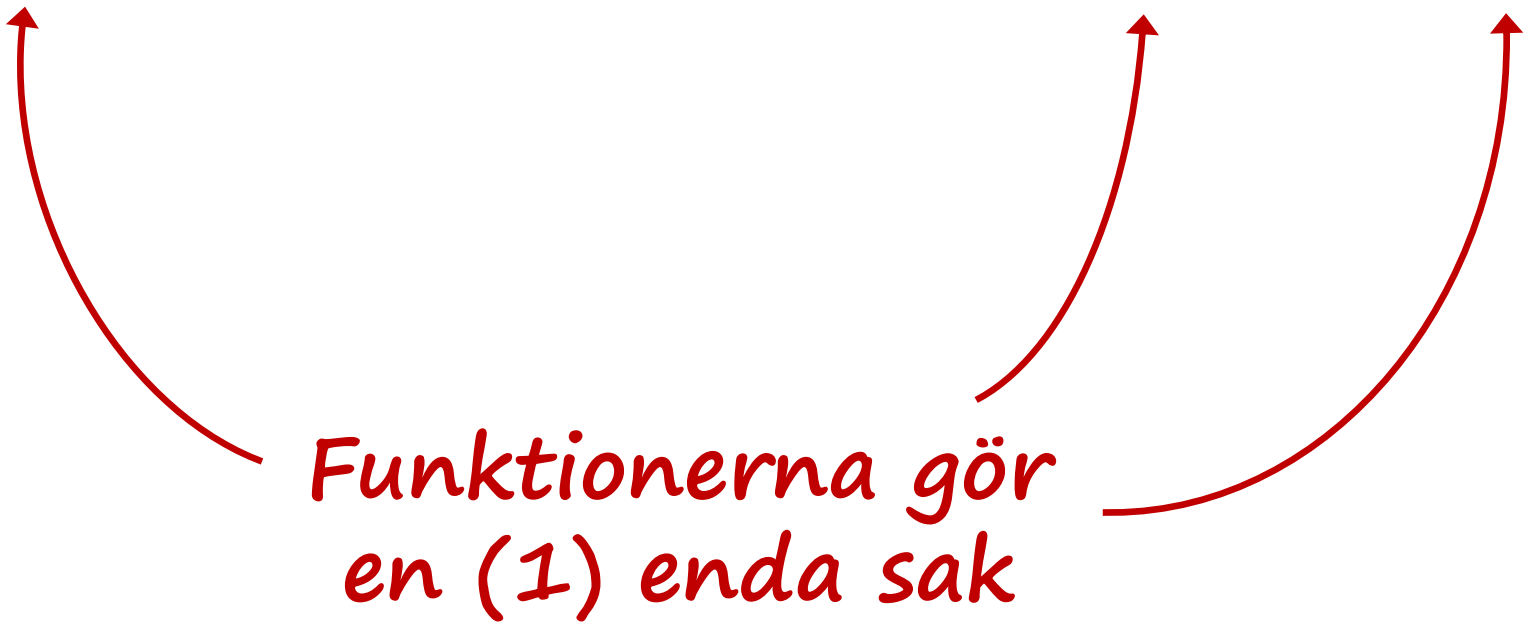
Higher-order function (HOF)



Exempelkod från https://en.wikipedia.org/wiki/Function_composition_(computer_science)

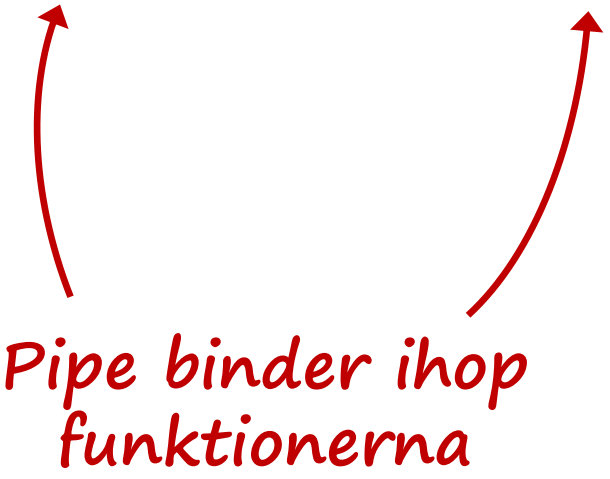
Komposition i Unix

```
find . -name "*.cs" | xargs cat | wc -l
```



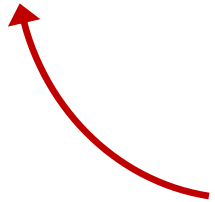
Komposition i Unix

```
find . -name "*.cs" | xargs cat | wc -l
```



Komposition ad hoc i C#

```
var result = FindFiles("*.cs")
                .Map(GetFileContent)
                .Map(CountLinesInContent);
```



I FP kallas pipe:n ofta
Bind *eller* Map
(motsvarar .Select från LINQ)

Problem: NullReferenceException

```
User user = users.GetUser(25);

Console.WriteLine(user.Name);
```

Lösning 1: if null

```
User user = users.GetUser(25);

if (user != null)
{
    Console.WriteLine(user.Name);
}
```

Lösning 2: Null-conditional operator

```
User user = users.GetUser(25);

Console.WriteLine(user?.Name);
```

Lösning 3: TryGet

```
if (users.TryGetUser(25, out var user))
{
    Console.WriteLine(user.Name);
}
```

Lösning 4: Tuple

```
(bool isFound, User user) = users.GetUser(25);

if (isFound)
{
    Console.WriteLine(user.Name);
}
```

Lösning 5: Exists property

```
User user = users.GetUser(25);

if (user.Exists)
{
    Console.WriteLine(user.Name);
}
```

Lösning 6: Result class (decorator)

```
UserResult result = users.GetUser(25);

if (result.Exists)
{
    Console.WriteLine(result.User.Name);
}
```

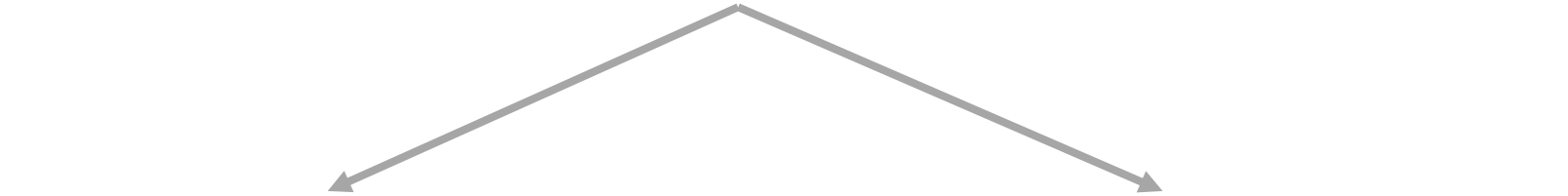
Lösning 7: Monad (FP)

```
Option<User> user = users.GetUser(25);

user.Map(x => Console.WriteLine(x.Name));
```

Även kallad Maybe

Option<User>



None

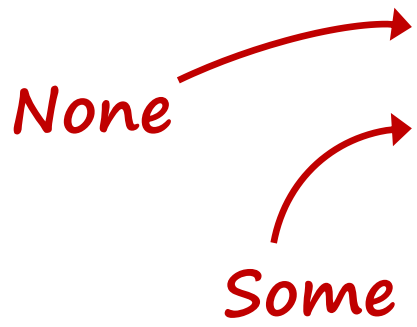
(null)

Some

Option<T>.Match

```
Option<User> user = users.GetUser(25);


string result = user.Match(
                    () => "Ingen användare hittad",
                    (user) => $"Hej {user.Name}!");
```



Hur man skapar Option<T>

```
public Option<User> GetUser(int id)
{
    var data = Database.Get("SELECT * FROM users WHERE id = @id", id);

    if (data != null)
    {
        return new User() { name = data["name"] };
    }

    return null;
}
```

Implicit konvertering till Some

Implicit konvertering till None

```
93          => Apply(@this.Map(F.CurryFirst), arg);
94
95      public static Option<Func<T2, T3, T4, R>> Apply<T1, T2, T3, T4, R>
96          (this Option<Func<T1, T2, T3, T4, R>> @this, Option<T1> arg)
97          => Apply(@this.Map(F.CurryFirst), arg);
98
99      public static Option<Func<T2, T3, T4, T5, R>> Apply<T1, T2, T3, T4, T5, R>
100         (this Option<Func<T1, T2, T3, T4, T5, R>> @this, Option<T1> arg)
101         => Apply(@this.Map(F.CurryFirst), arg);
102
103     public static Option<Func<T2, T3, T4, T5, T6, R>> Apply<T1, T2, T3, T4, T5, T6, R>
104         (this Option<Func<T1, T2, T3, T4, T5, T6, R>> @this, Option<T1> arg)
105         => Apply(@this.Map(F.CurryFirst), arg);
106
107     public static Option<Func<T2, T3, T4, T5, T6, T7, R>> Apply<T1, T2, T3, T4, T5, T6, T7, R>
108         (this Option<Func<T1, T2, T3, T4, T5, T6, T7, R>> @this, Option<T1> arg)
109         => Apply(@this.Map(F.CurryFirst), arg);
110
111     public static Option<Func<T2, T3, T4, T5, T6, T7, T8, R>> Apply<T1, T2, T3, T4, T5, T6, T7, T8, R>
112         (this Option<Func<T1, T2, T3, T4, T5, T6, T7, T8, R>> @this, Option<T1> arg)
113         => Apply(@this.Map(F.CurryFirst), arg);
114
115     public static Option<Func<T2, T3, T4, T5, T6, T7, T8, T9, R>> Apply<T1, T2, T3, T4, T5, T6, T7, T8, T9, R>
116         (this Option<Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, R>> @this, Option<T1> arg)
117         => Apply(@this.Map(F.CurryFirst), arg);
118
119     public static Option<R> Bind<T, R>
120         (this Option<T> optT, Func<T, Option<R>> f)
121          => optT.Match(
122             () => None,
123             (t) => f(t));
```

Notera Match

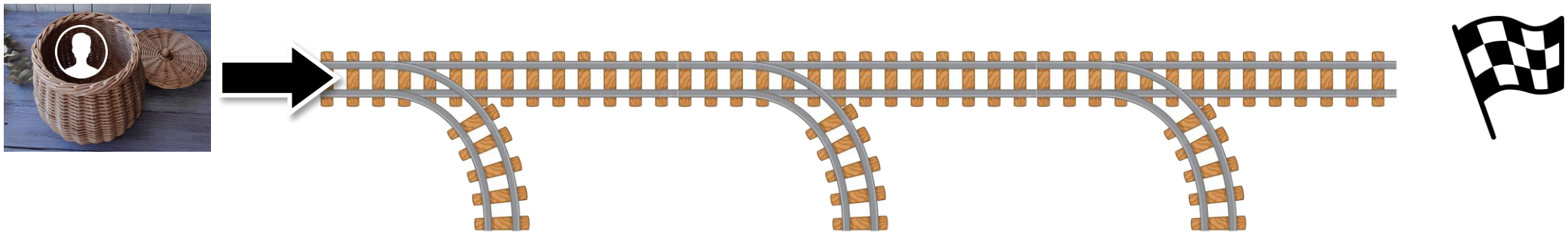
Hög abstraktion

Källa: https://github.com/la-yumba/functional-csharp-code/blob/master/LaYumba.Functional/Option.cs

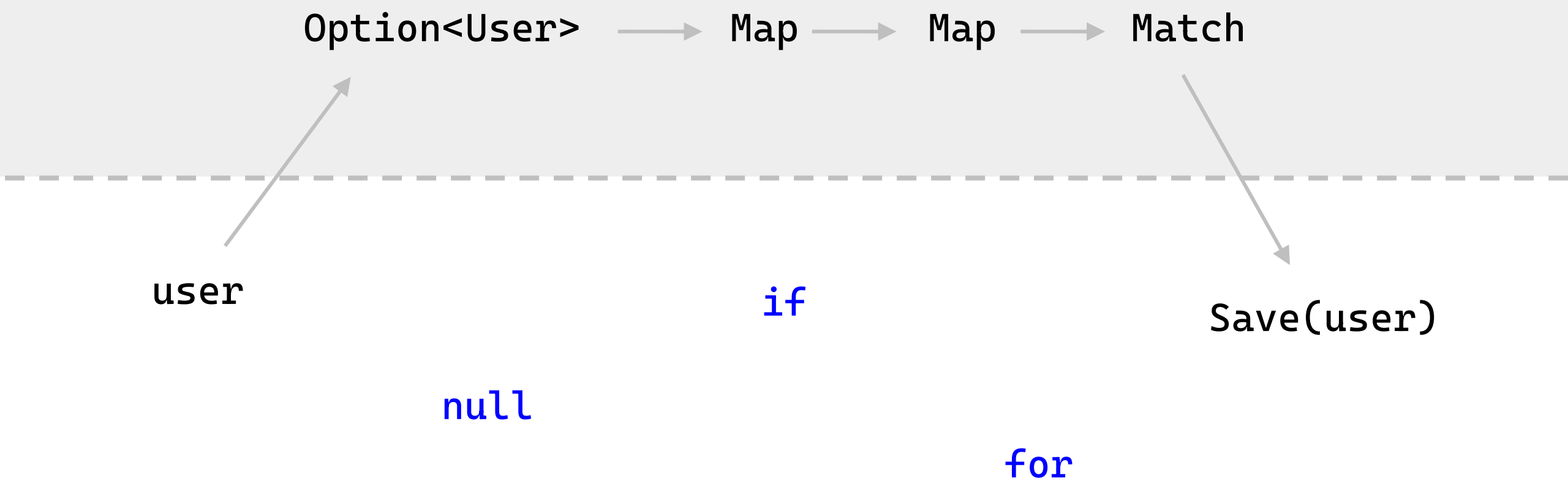
Option + Map

```
Option<User> user = users.GetUser(25);

user
    .Map(CalculateUserExpireDate)
    .Map(IsUserExpired)
    .Map(SendEmailToAdmin);
```

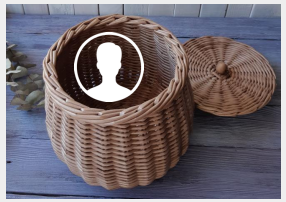


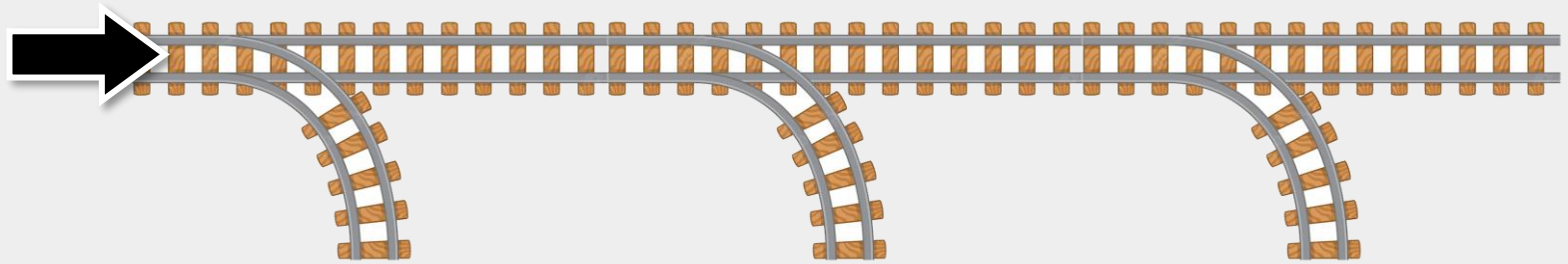
Den ”upplyfta” världen



Den vanliga världen

Den ”upplyfta” världen





`user`

Målet är att stanna i den upplyfta världen så länge som möjligt & bygga pipelines med dataflöden

`Save(user)`

Den vanliga världen

Monader i C#

`Task<T>`

`Lazy<T>`

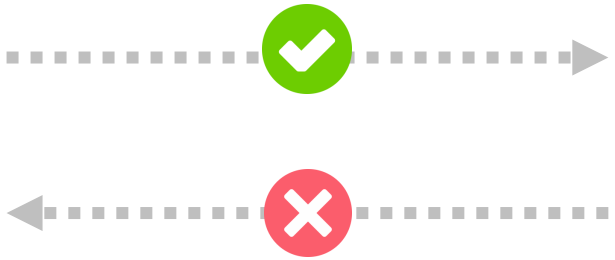
`await`

Me Introducing My Coworkers to Functional Programming

Tips för att skriva FP i C#

Försök separera allt till en av två världar

Orena void-metoder med bieffekter



Rena funktioner utan bieffekter som returnerar data

Tips för att skriva FP i C#

Undvik states, returnera nytt värde (immutable)

```
int number = 4;
number++;
```

```
int number = 4;
int newNumber = number + 1;
```

Tips för att skriva FP i C#

Undvik states, returnera nytt värde (immutable)

```
public class Person
{
    public Person(string name, int age)
    {
        // Validering här
        Name = name;
        Age = age;
    }
    public string Name { get; }
    public int Age { get; }
}
```

Inget set

Tips för att skriva FP i C#

Undvik states, returnera nytt värde (immutable)

```
public class Person
{
    public Person(string name, int age)
    {
        // Validering här
        Name = name;
        Age = age;
    }
    public string Name { get; }
    public int Age { get; }
}
```

Tips för att skriva FP i C#

Undvik states, returnera nytt värde (immutable)

```
public class Person
{
    public Person(string name, Age age)
    {
        // Validering här
        Name = name;
        Age = age;
    }
    public string Name { get; }
    public Age Age { get; }
}
```

Tips för att skriva FP i C#

Undvik states, returnera nytt värde (immutable)

```
public class Age
{
    public Age(int age)
    {
        // Validering här
        Age = age;
    }

    public int Value { get; }
}
```

Tips för att skriva FP i C#

Undvik states, returnera nytt värde (immutable)

```
public record Person(string Name, int Age);

var chuck = new Person("Chuck Norris", 82);
var younger = chuck with { Age = 50 };
```

Tips för att skriva FP i C#

Använd `static` för att undvika states

```
public static class Utils
```

Tips för att skriva FP i C#

Låt funktioner göra en (1) sak.

Separera beteende från data.

```
public class Person
{
  public string Name { get; set; }
  public int Age { get; set; }

  public void Save() { }
  public void Update() { }
  public List<Person> GetPersons() { }

  public void CalcExpireDate() { }
}
```

```
public record Person(string Name, int Age);


public class PersonService
{
  public void Save(Person p) { }
  public void Update(Person p) { }
  public List<Person> GetList() { }
}

public static class DateCalculator
{
  public DateTime CalcExpireDate(Person p) { }
}
```

Tips för att skriva FP i C#

Låt funktioner vara data

```
T GetSomething(string name)
{
    if (name == "A")
        return new A();
    else if (name == "B")
        return new B();
    else if (name == "C")
        return new C();
}
```

```
var classes = new Dictionary<string, Func<T>>()
{
    {"A", () => new A() },
    {"B", () => new B() },
    {"C", () => new C() },
};

classes[name].Invoke();
```

Tips för att skriva FP i C#

`Interface` bör används för olika implementationer – inte för att bara köra godtycklig funktion. Använd `Func` & `Action` i stället.

```
public interface ITextWriter
{
   void Write(string text);
}

public class TextWriter : ITextWriter
{
   public void Write(string text) => Console.WriteLine(text);
}

public class WriteSomeText
{
   private ITextWriter _writer;

   public WriteSomeText(ITextWriter writer)
   {
      _writer = writer;
   }

   public void Write(string text) => _writer.Write(text);
}

var writer = new WriteSomeText(new TextWriter());
writer.Write("Hej!");
```

```
public static class WriteSomeText
{
   public static void Write(string text, Action<string> method)
     => method(text);
}

WriteSomeText.Write("Hej!", x => Console.WriteLine(x));
```

Tips för att skriva FP i C#

Samma datatyp för indata och returdata – kedja metoder

```
string text = "Hej";

text
  .ToUpper()
  .ToLower()
  .Normalize()
  .ForEach(Console.WriteLine);
```

Notera att bieffekterna sker sist i kedjan

Tips för att skriva FP i C#

Lyft ut boiler plate-kod till egna funktioner

```
public long TimeIt(Action method)
{
    var sw = new Stopwatch();
    sw.Start();
    method();
    sw.Stop();
    return sw.Elapsed.TotalMilliseconds;
}

long millisecs = TimeIt(() => MyExpensiveFunction());
```

Sammanfattning

OOP

(imperativ)

FP

Sammanfattning

OOP	FP
Objekt	Funktioner
State	Stateless
Mutable	Immutable
Side effects (unpure)	No side effects (pure)
Inheritance	Composition

Funktioner är också data!

Använd både OOP och FP

Saker jag inte tagit upp

- Currying
- Partial applications
- Closures
- Apply

- Functors
- Applicatives (applicative functors)
- Monoid

- Fold/reduce (motsvarar LINQ Aggregate – Sum är ett exempel på aggregate)

Läs mer

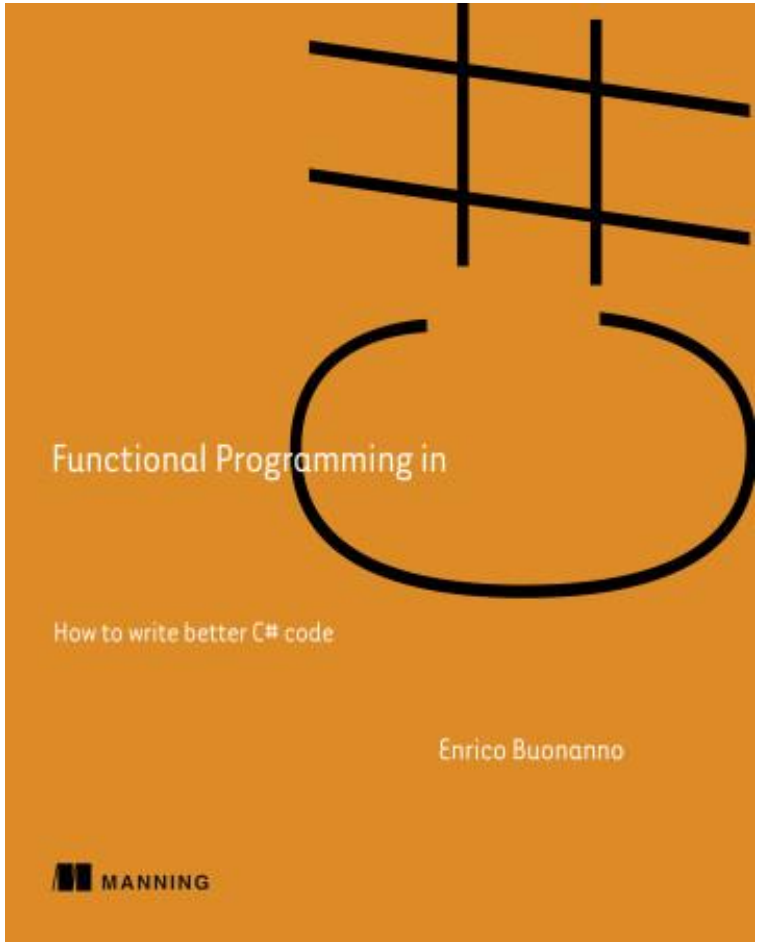
Bloggar
Scott Wlaschin fsharpforfunandprofit.com
Mark Seemann blog.ploeh.dk

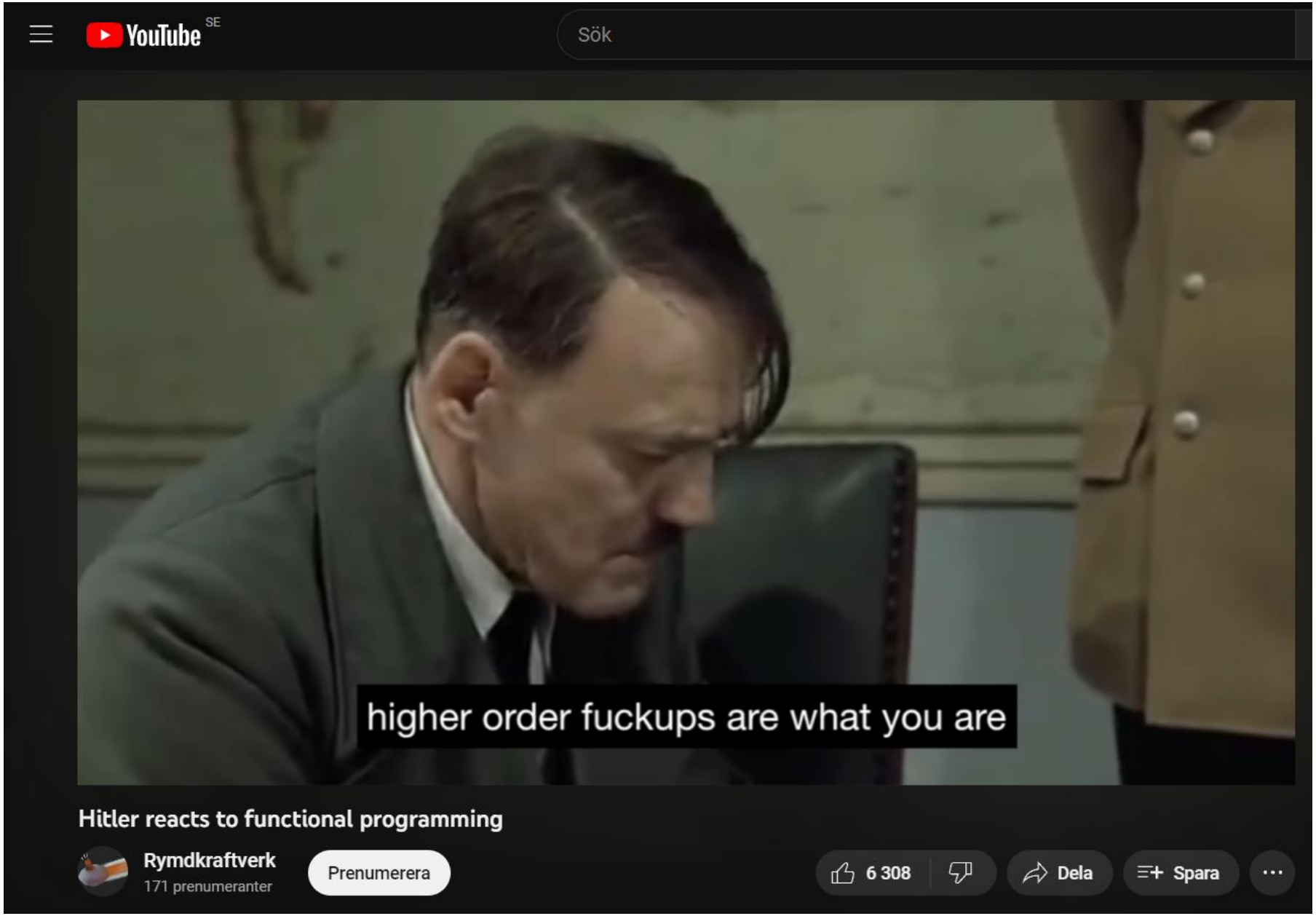
Böcker
Enrico Buonanno (2021) Functional Programming in C#, 2nd ed

YouTube
Get value out of your monad - Mark Seemann
https://www.youtube.com/watch?v=F9bznonKc64

NuGet package
https://github.com/louthy/language-ext

Intressanta design patterns
- Memoization
- Monad
- Map/Reduce/Filter (motsvarar LINQ Select/Aggregate/Where)





https://www.youtube.com/watch?v=ADqLBc1vFwI