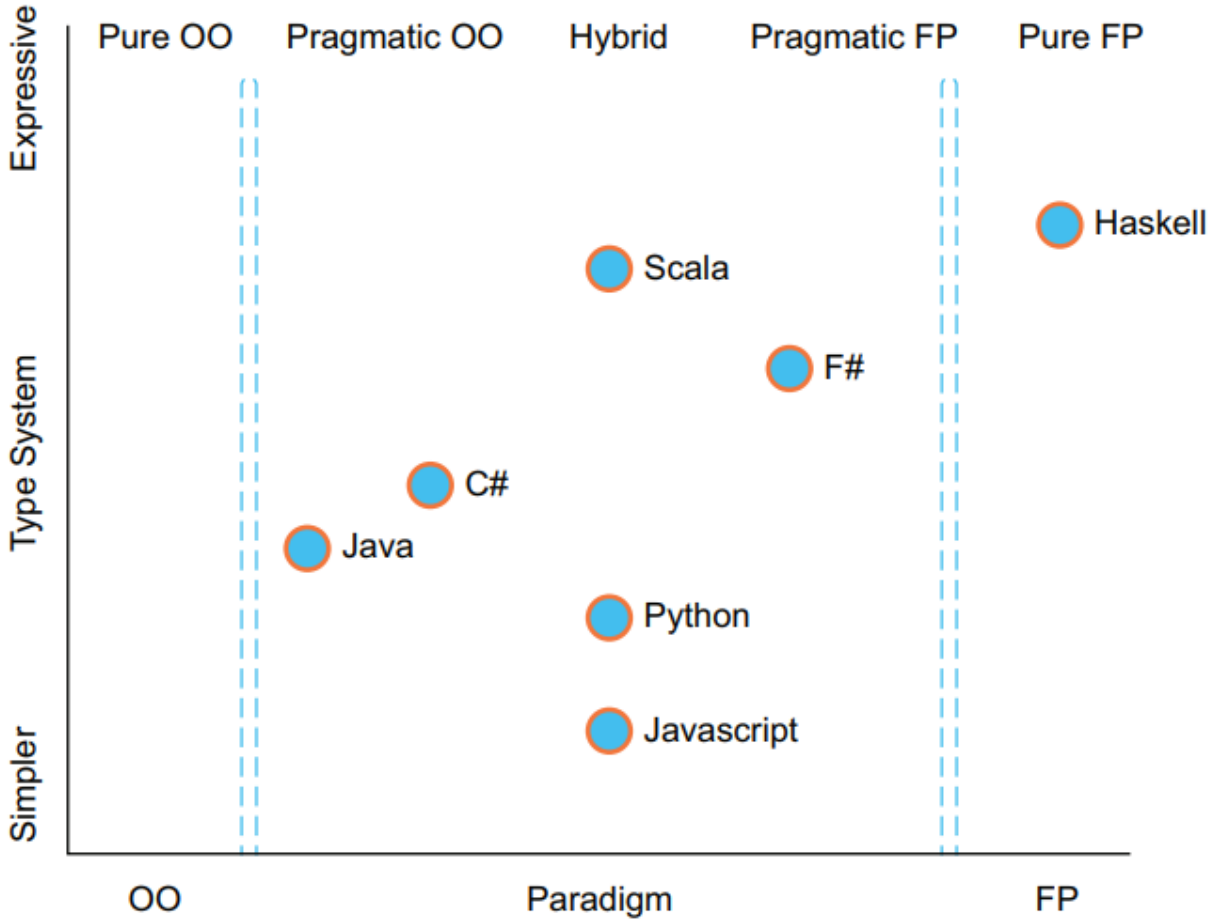
You |> learn F#

Bartłomiej Soter
Łukasz Krzywizna

About language

F#

- Open source,
- Cross-platform
- Statically typed
- Functional (based on Ocaml)
- But also object-oriented (cause leverage .NET runtime)
- With REPL support



Source: Get Programming with F#

REPL (read-eval-print loop)

Language shell that allows for:

- Taking user input:
- Evaluate functions and variables
- Error handling
- Check declared statement

Use it by executing (in terminal):

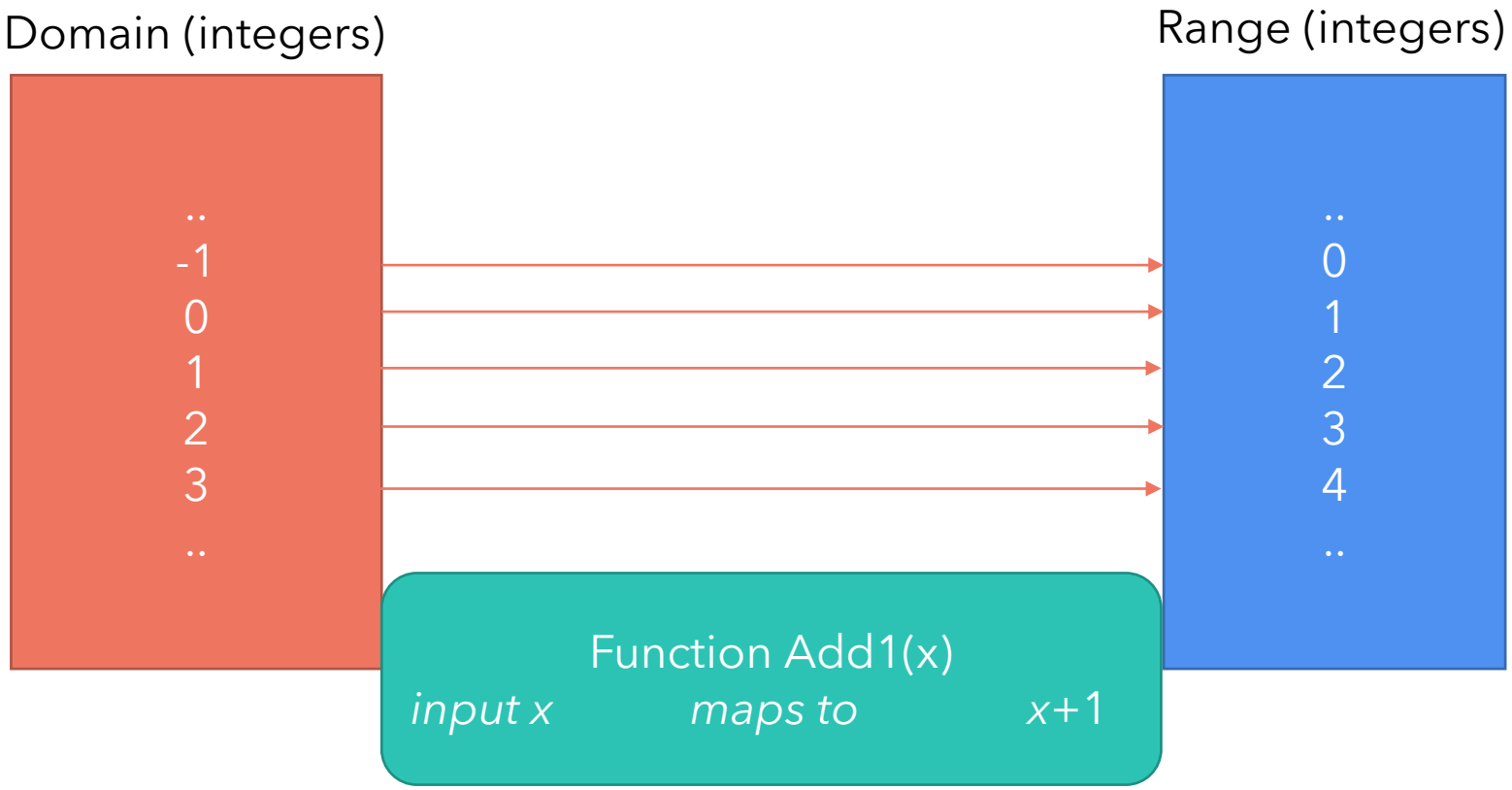
dotnet fsi

Functional thinking

Main concepts

- There are **values** and **functions** (really – nothing more)
- Values are **immutable**
- No exceptions but **Result** instead
- Avoid nulls with **Option**
- Functions are **pure**:
  - can't produce side effect
  - for the same input gives same output
- And have first-class status:
  - Have name
  - Can be passed to other function/structure
  - Can be returned

Main concepts



Source: fsharpforfunandprofit.com

Power of pure functions

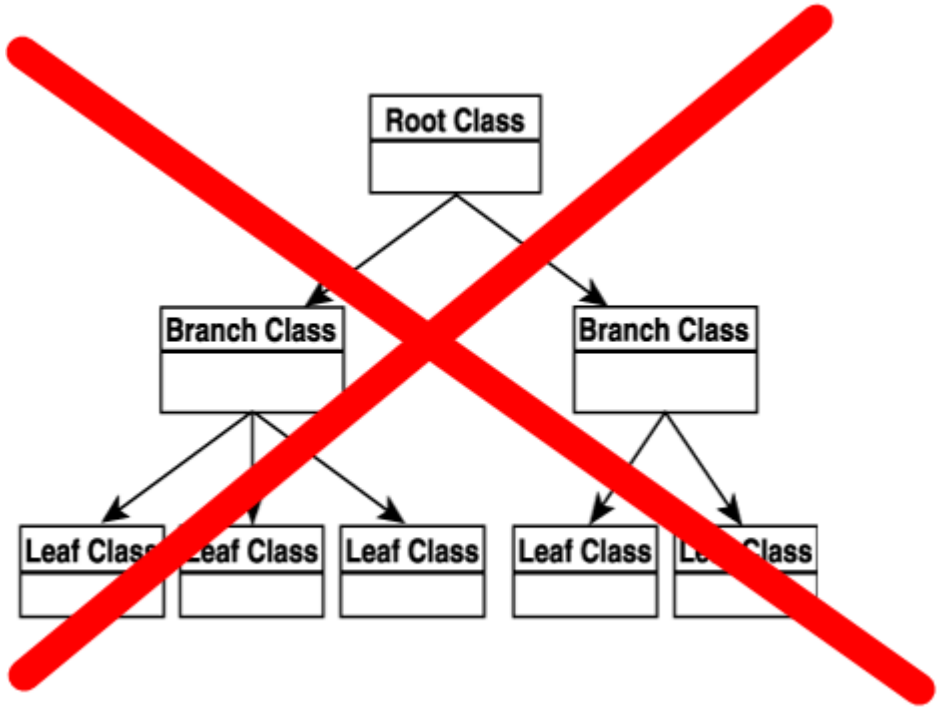
- Trivially parallelizable
- Can be used lazily – only evaluate when I need it. Now or later
- Easily caching
- Pure functions can be use together in any order vs OOP when executing functions manipulate object state

Immutability

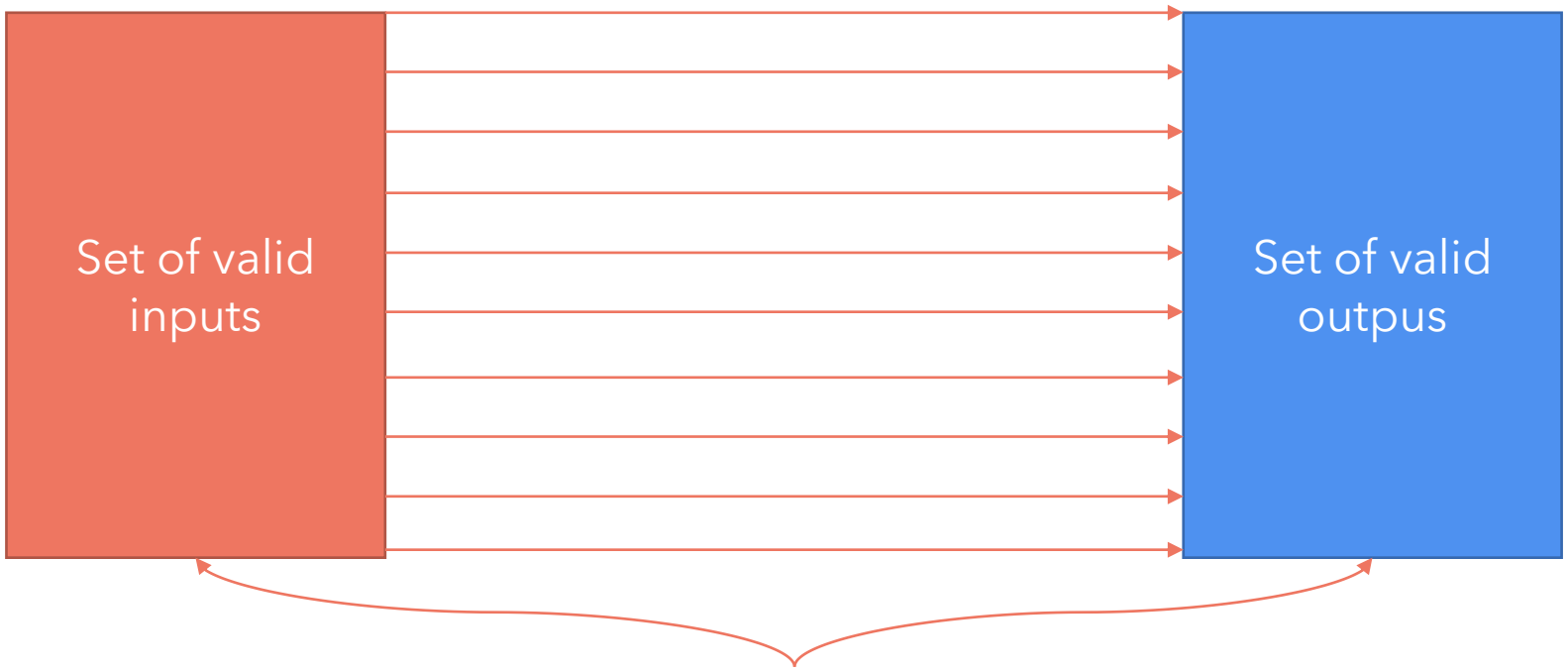
- Immutable data makes the code predictable
- Immutable data is easier to work with – I'm sure that data passed to function is safe
- Immutable data forces you to use a "transformational" approach – look at the SQL or LINQ

Types

Types are not classes



Types



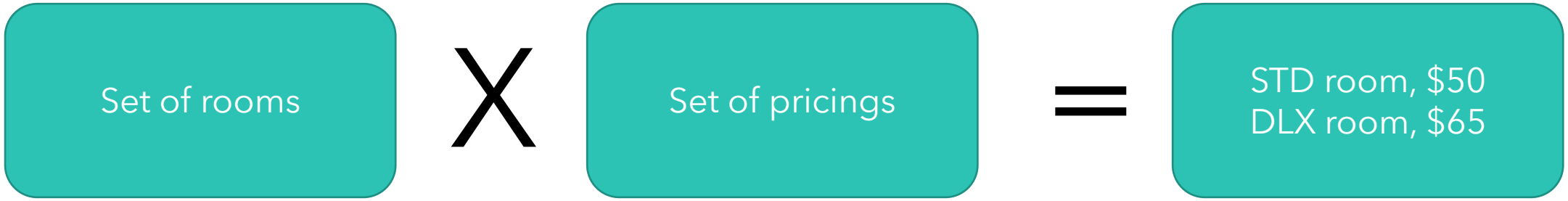
There are just labels for a set of inputs/outputs
Int is type, **string** is type

Type separate data from behavior

Source: fsharpforfunandprofit.com

Composing Types

Product type:



type Offer = Room * Pricing

Source: fsharpforfunandprofit.com

Composing Types

Sum type:

Set of individuals

+

Set of travel agents

+

Set of companies

```
type ReservationHolder =
| Individual
| TravelAgent
| Company
```

Source: fsharpforfunandprofit.com

Type signature

Most functional languages use "arrowed" type signature for a function:

```
int -> int
```

Input

Output

Type signature

Most functional languages use "arrowed" type signature for a function:

```
int -> int -> int
```

Input

Output

Type signature

Most functional languages use "arrowed" type signature for a function:

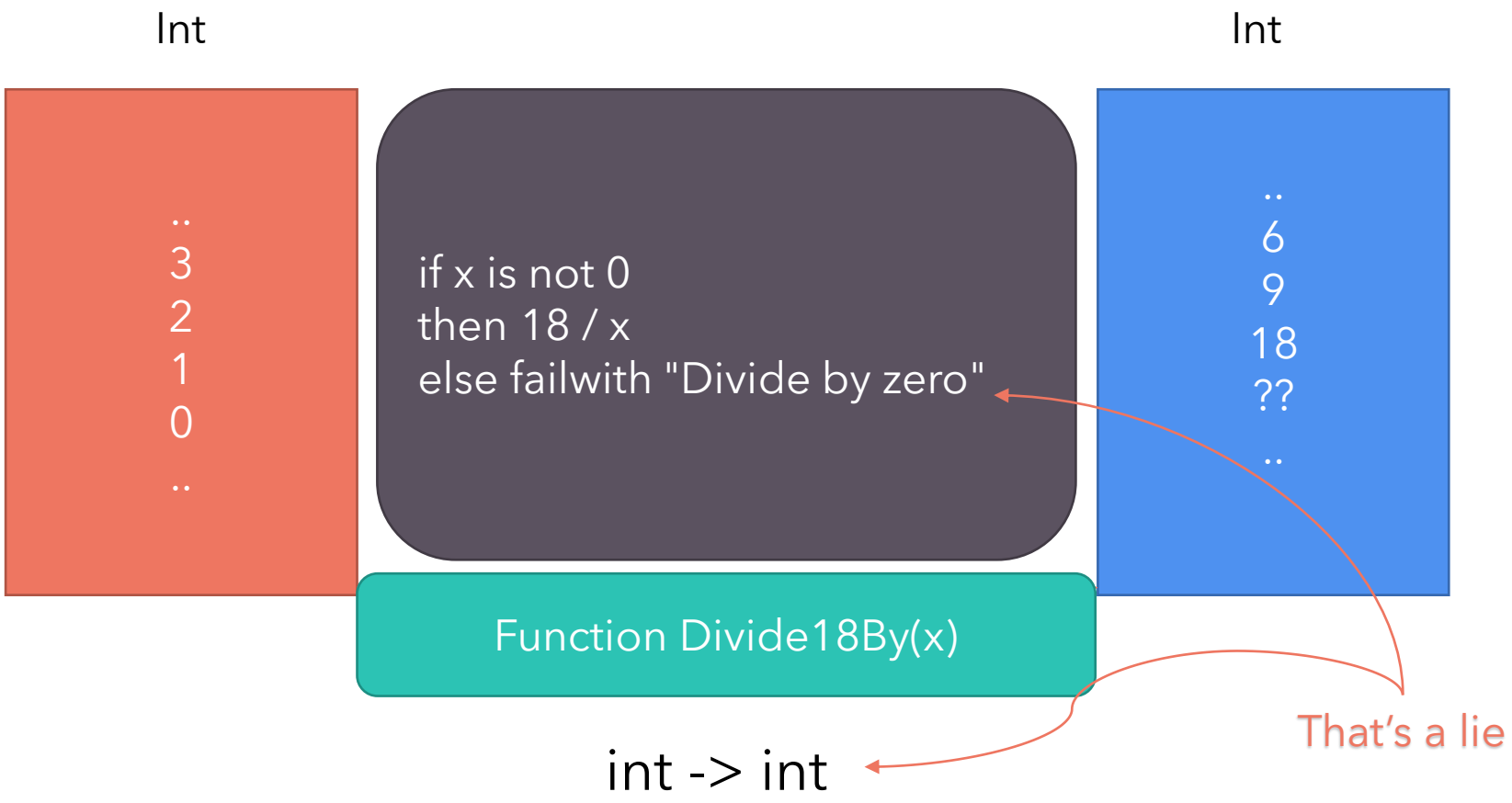
```
int -> (int -> string) -> string
```

Input

Also input: another function

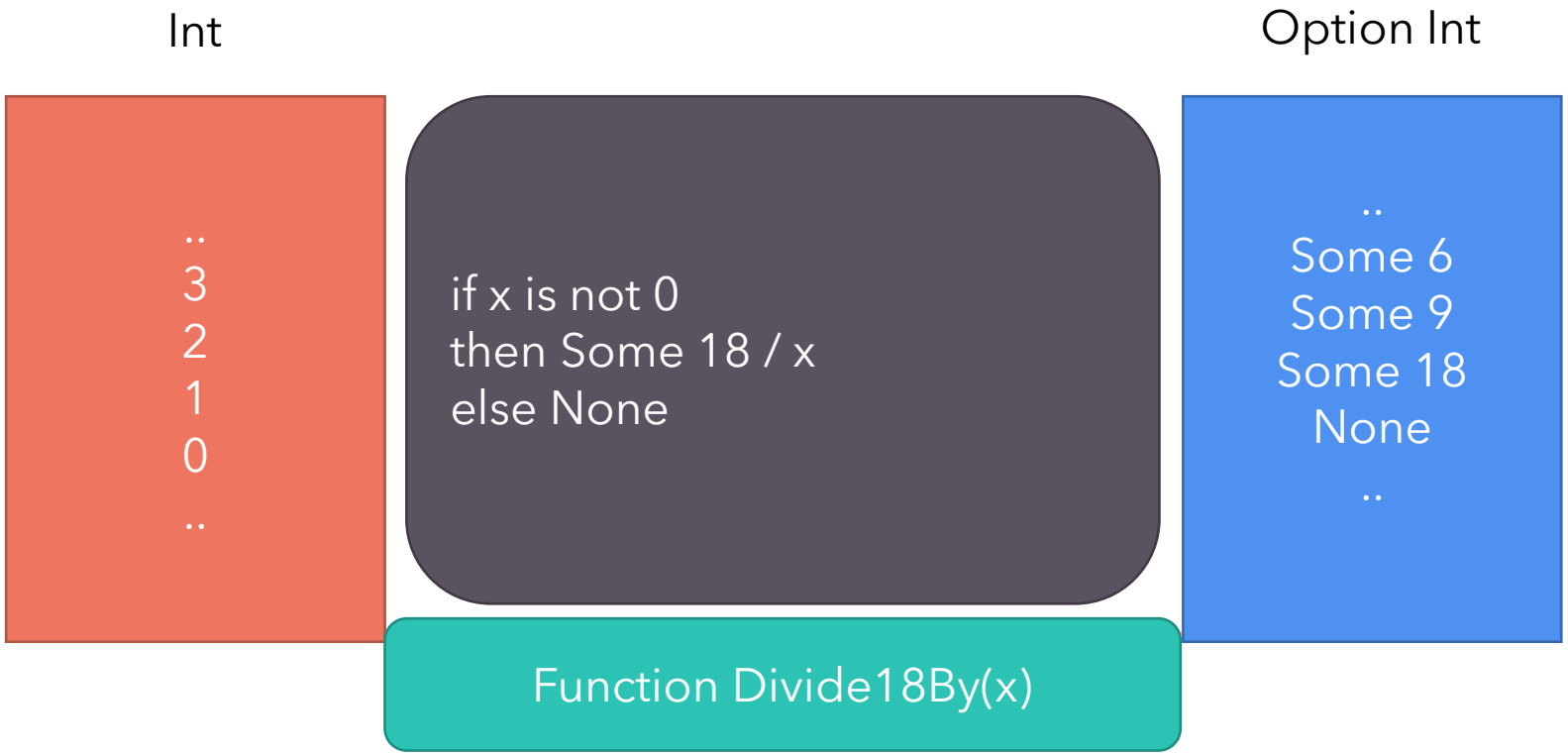
Output

Option



Source: fsharpforfunandprofit.com

Option



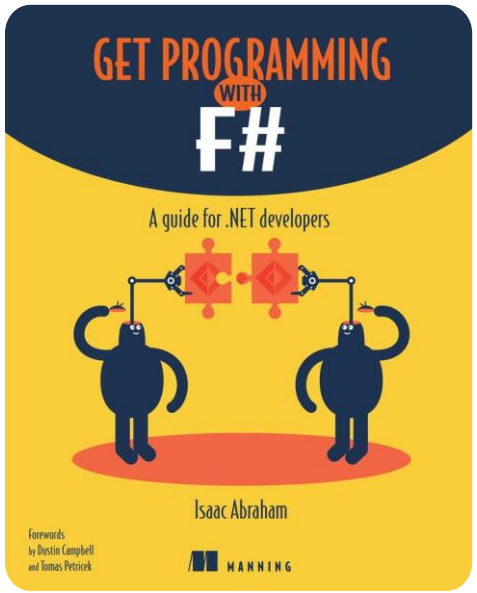
int -> int option

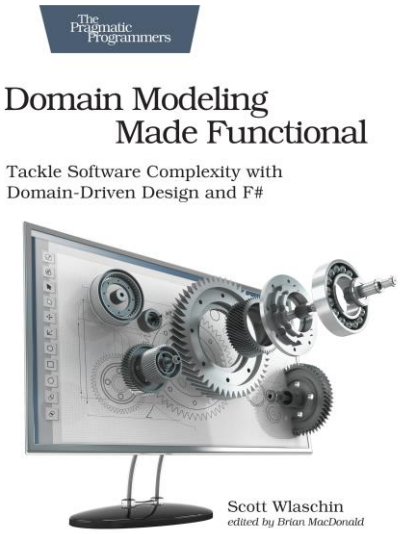
Source: fsharpforfunandprofit.com

Let’s see some code

Presentation based on:

- https://fsharpforfunandprofit.com/
- Learn F# | Free tutorials, courses, videos, and more | .NET (microsoft.com)
- F# Software Foundation (fsharp.org)
- F# on Twitter Discussion Group
- Get Programming with F# (manning.com)
- Domain Modeling Made Functional: Tackle Software Complexity with Domain-Driven Design and F# by Scott Wlaschin (pragprog.com)





Thank you

Bartłomiej Soter

https://github.com/jabarij

Łukasz Krzywizna

https://github.com/lukaszkrzywizna