

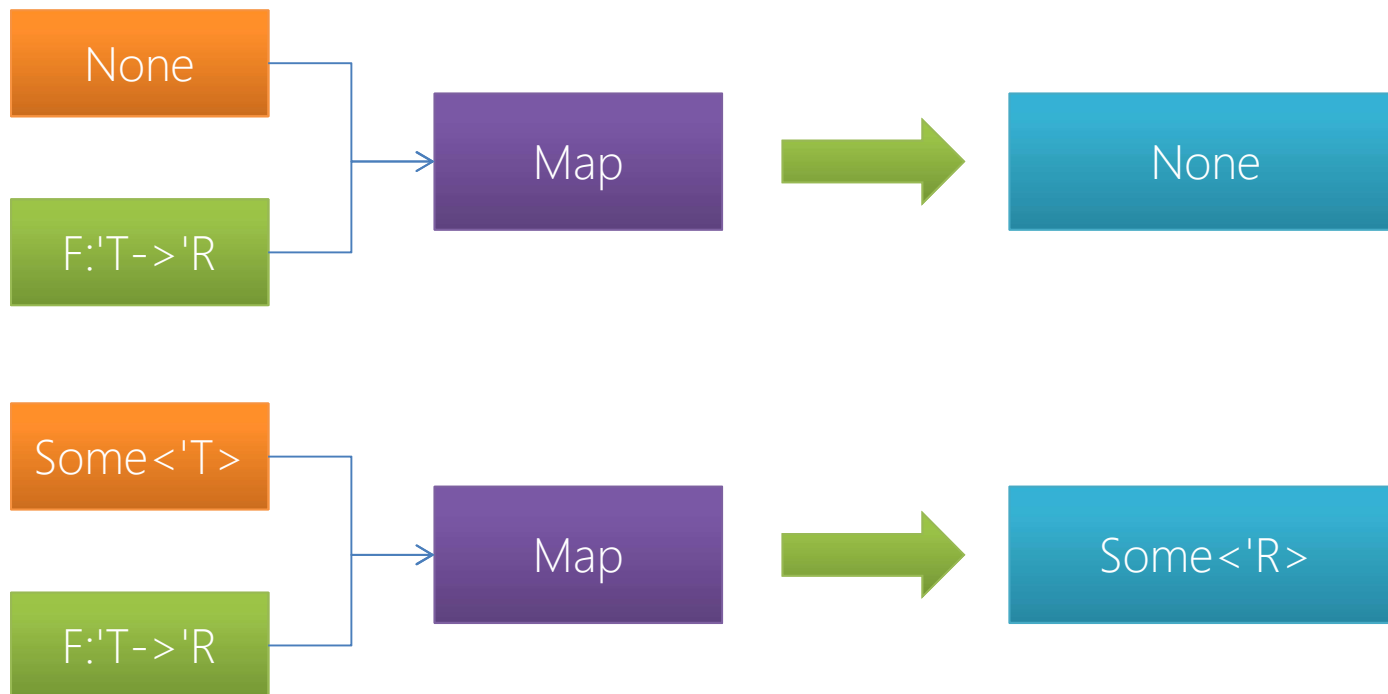
MAPOWANIE, FUNKTORY I MONADY

Mapowanie

mapuj: $M <' T > \rightarrow ('T \rightarrow 'R) \rightarrow M <' R >$

mapuj: $\text{option} <' T > \rightarrow ('T \rightarrow 'R) \rightarrow \text{option} <' R >$

Mapowanie



Mapowanie

```
let mapuj mapa opcja =  
  match opcja with  
  | Some x -> Some (mapa x)  
  | None -> None
```

Funktory

$\text{mapuj: } M \langle 'T \rangle \rightarrow ('T \rightarrow 'R) \rightarrow M \langle 'R \rangle$

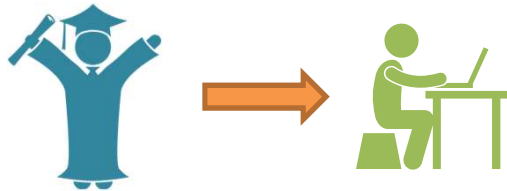
W programowaniu funkcyjnym dowolny typ wyposażony
w funkcję mapuj nazywamy funktorem

Prawa funktorów

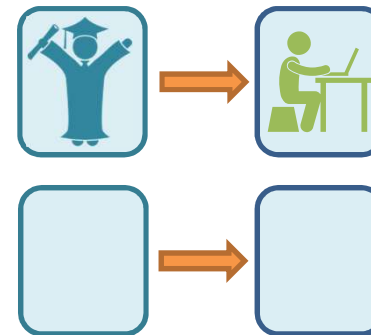
Mapowanie musi zachowywać strukturę kontenera

Kandydat: $\text{Option} < \text{Student} >$

uczelnia: $\text{Student} \rightarrow \text{Inżynier}$



mapuj uczelnia: $\text{Option} < \text{Student} > \rightarrow \text{Option} < \text{Inżynier} >$



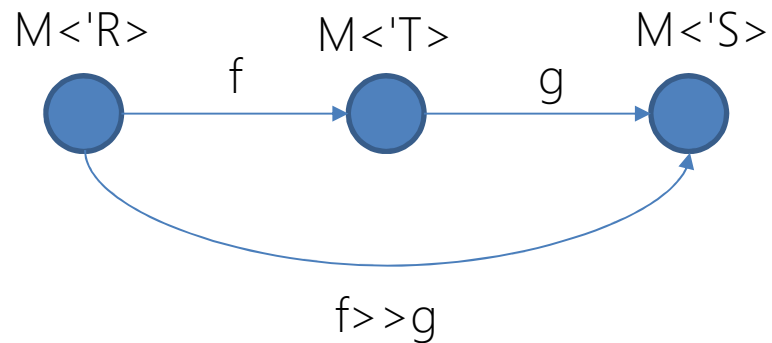
Prawa funktorów

Mapowanie musi zachowywać strukturę kontenera

1. `mapuj id = id`

id: $x \rightarrow x$

2. `mapuj (f >> g) = mapuj f >> mapuj g`



Prawa funktorów - opcje

```
let id x = x
```

```
let mapujId = mapuj id
```

```
(mapujId (Some x)) = (Some x)
```

```
(id (Some x)) = (Some x)
```

```
let mapujId opcja =  
  match opcja with  
  | Some x -> Some (id x)  
  | None -> None
```

```
(mapujId None) = None
```

```
(id None) = None
```

```
mapuj id == id
```


Prawa funktorów - opcje

$(\text{mapuj } f \text{ (Some } x)) = (\text{Some } (f \ x))$

$(\text{mapuj } f \text{ None}) = \text{None}$

$(\text{mapuj } g \text{ (Some } (f \ x)) = (\text{Some } (g \ (f \ x)))$

$(\text{mapuj } g \text{ None}) = \text{None}$

$(\text{mapuj } (f \gg g) \text{ (Some } x)) = (\text{Some } (f \gg g \ x))$

$(\text{mapuj } (f \gg g) \text{ None}) = \text{None}$

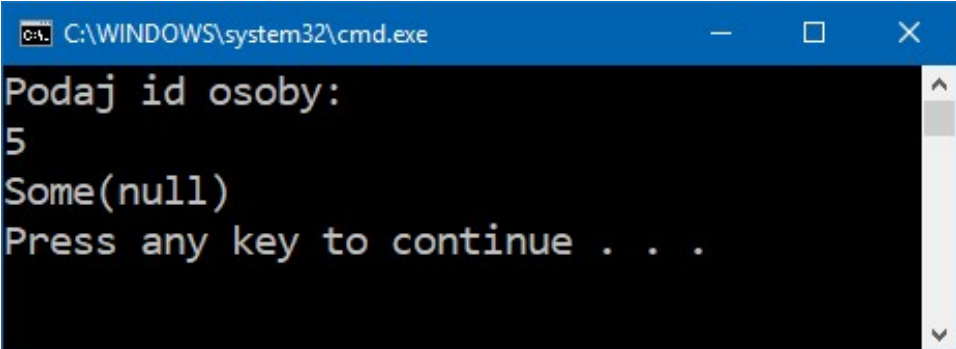
Mapowania

```
let naString (osoba:Osoba) =  
    sprintf "%d\n%s\n%s" osoba.id osoba.imie osoba.nazwisko
```

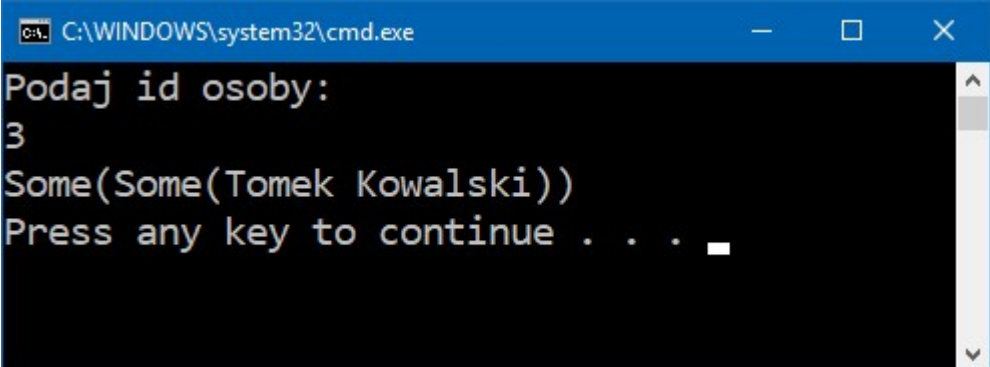
```
[<EntryPoint>]  
let main argv =  
    wczytajDane "dane.json"  
    |> ... (fun dane->  
        loop wczytajId  
        |> znajdzOsobePoId dane  
        |> mapuj naString  
    )  
    |> printfn "%A"  
    0
```

Mapowania

```
[<EntryPoint>]
let main argv =
    wczytajDane "dane.json"
    |> mapuj (fun dane->
        loop wczytajId
        |> znajdzOsobePoId dane
        |> mapuj --> Główny
    )
    |> printfn "%A"
0
```



```
C:\WINDOWS\system32\cmd.exe
Podaj id osoby:
5
Some(null)
Press any key to continue . . .
```



```
C:\WINDOWS\system32\cmd.exe
Podaj id osoby:
3
Some(Some(Tomek Kowalski))
Press any key to continue . . .
```

Funktory

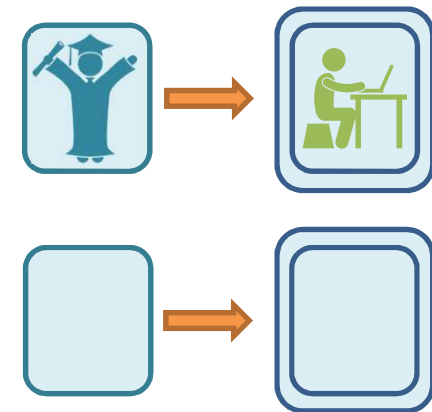
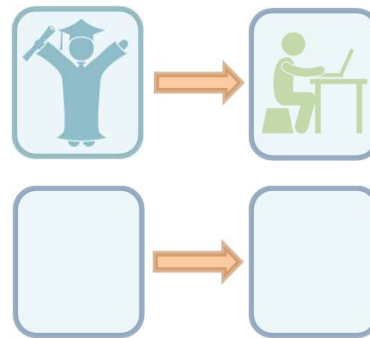
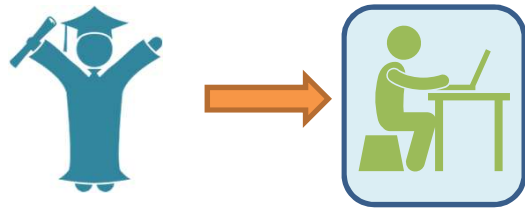
Czy funkcja uczelnia jest poprawnie zdefiniowana?

Kandydat: $\text{Option} < \text{Student} >$

uczelnia: $\text{Student} \rightarrow \text{Opcja} < \text{Inżynier} >$

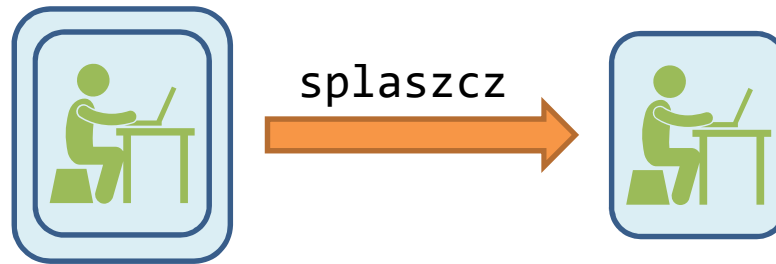
mapuj uczelnia: $\text{Option} < \text{Student} >$

$\rightarrow \text{Option} < \text{Option} < \text{Inżynier} >$



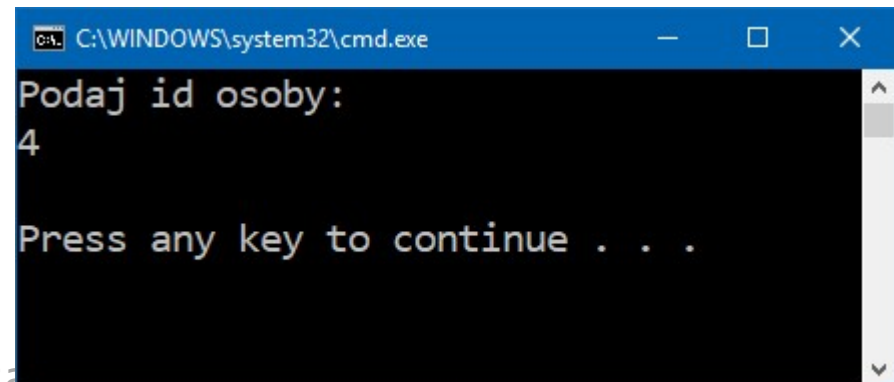
Spłaszczanie opcji

```
let splaszcz opcja =  
  match opcja with  
  | Some (Some x) -> Some x  
  | None -> None
```

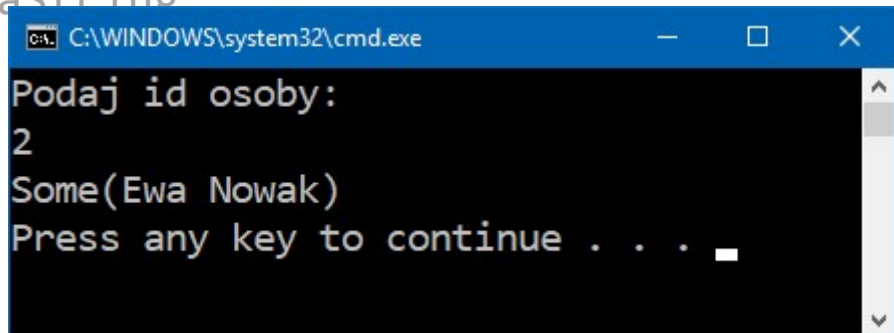


Spłaszczanie opcji

```
[<EntryPoint>]
let main argv =
    wczytajDane "dane.json"
    |> mapuj (fun dane->
        loop wczytajDane "dane.json"
        |> znajdzOsobePoId dane
        |> mapuj naString
    )
    |> splaszcz
    |> printfn "%A"
0
```



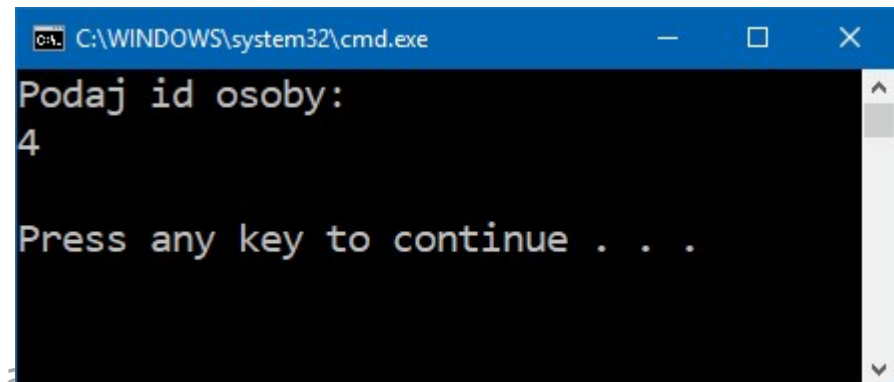
```
C:\WINDOWS\system32\cmd.exe
Podaj id osoby:
4
Press any key to continue . . .
```



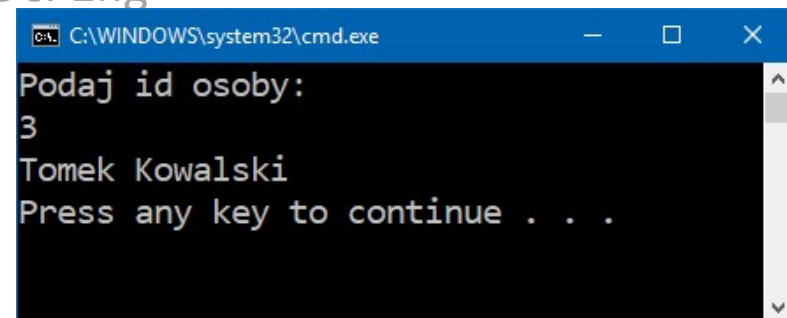
```
C:\WINDOWS\system32\cmd.exe
Podaj id osoby:
2
Some(Ewa Nowak)
Press any key to continue . . .
```

Opcje

```
[<EntryPoint>]
let main argv =
    wczytajDane "dane.json"
    |> mapuj (fun dane->
        loop wczytajDane "dane.json"
        |> znajdzOsobePoId dane
        |> mapuj naString
    )
    |> splaszczy
    |> wykonaj (printfn "%s")
0
```



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window has a black background with white text. It displays the prompt "Podaj id osoby:" followed by the user input "4". Below the input, it says "Press any key to continue . . .".



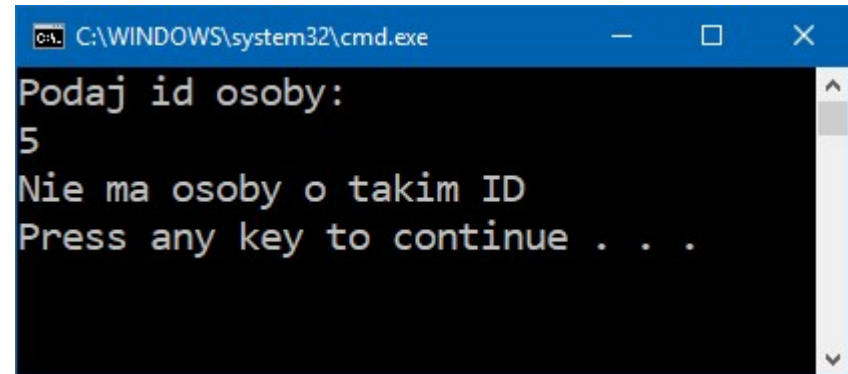
A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window has a black background with white text. It displays the prompt "Podaj id osoby:" followed by the user input "3". Below the input, it shows the output "Tomek Kowalski" and then "Press any key to continue . . .".

Opcje

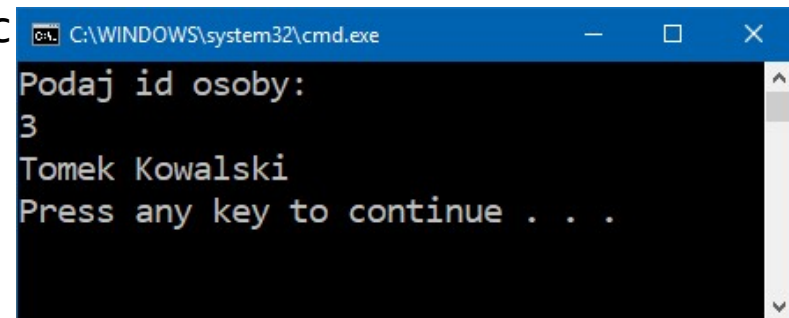
```
let gdyNieMaWartosci zastepcza opcja =  
  match opcja with  
  | Some x -> Some x  
  | None -> Some zastepcza
```


Mapowania

```
[<EntryPoint>]
let main argv =
    wczytajDane "dane.json"
    |> mapuj (fun dane->
        loop wczytajId
        |> znajdzOsobePoId dane
        |> mapuj naString
        |> gdyNieMaWartosc
    )
    |> splaszcz
    |> wykonaj (printfn "%A")
0
```



```
C:\WINDOWS\system32\cmd.exe
Podaj id osoby:
5
Nie ma osoby o takim ID
Press any key to continue . . .
```



```
C:\WINDOWS\system32\cmd.exe
Podaj id osoby:
3
Tomek Kowalski
Press any key to continue . . .
```

Bindowanie

złącz: $M < 'T > \rightarrow ('T \rightarrow M < 'R >) \rightarrow M < 'R >$

złącz: $\text{option} < 'T > \rightarrow ('T \rightarrow \text{option} < 'R >) \rightarrow \text{option} < 'R >$

```
let złącz opcja mapa =  
    match opcja with  
    | Some x -> mapa x  
    | None -> None
```

Monady

zwroc: $T \rightarrow M < 'T >$

zwroc: $(T) \rightarrow \text{option} < 'T >$

W programowaniu funkcyjnym dowolny typ wyposażony
w funkcje `złącz` i `zwroc` nazywamy monadą

Prawa monad

1. Tożsamość lewostrona

`złacz zwroc(x) g = g(x)`

2. Tożsamość prawostronna

`złacz M(x) zwroc = M(x)`

3. Łączność

`złacz M(x) (fun x-> złacz f(x) g)
złacz (złacz M(x) f) g`

Sprawdzenie praw monad dla opcji (1)

```
let złącz opcja mapa =  
  match opcja with  
  | Some x -> mapa x  
  | None -> None
```

$g: 'A \rightarrow \text{Option} <'B>$

1. Tożsamość lewostrona

$\text{złącz} \text{ zwroc}(x) \ g = g(x)$

$\text{złącz} \text{ Some}(x) \ g \iff g(x)$

Sprawdzenie praw monad dla opcji (2)

```
let złącz opcja mapa =  
  match opcja with  
  | Some x -> mapa x  
  | None -> None
```

2. Tożsamość prawostronna

`złącz M(x) zwroc = M(x)`

`złącz Some(x) Some`  `Some(x)`

Sprawdzenie praw monad dla opcji (3)

```
let złącz opcja mapa =  
  match opcja with  
  | Some x -> mapa x  
  | None -> None
```

3. Łączność

$$\text{złącz } M(x) \text{ (fun } x \rightarrow \text{złącz } f(x) \text{ } g) = \text{złącz (złącz } M(x) \text{ } f) \text{ } g$$
$$\text{złącz } \text{Some}(x) \text{ (fun } x \rightarrow \text{złącz } f(x) \text{ } g) \Rightarrow \text{złącz } f(x) \text{ } g$$
$$\text{złącz (złącz } \text{Some}(x) \text{ } f) \text{ } g \Rightarrow \text{złącz } f(x) \text{ } g$$

Bindowanie

```
[<EntryPoint>]
let main argv =
    wczytajDane "dane.json"
    |> złącz (fun dane->
        loop wczytajId
        |> znajdzOsobePoId dane
        |> mapuj naString
        |> gdyNieMaWartosci "Nie ma osoby o takim ID"
    )
    |> wykonaj (printfn "%s")
0
```


Moduł Option

Operacja	Funkcja
Mapuj	Option.map
Złącz	Option.bind
Wykonaj	Option.iter
Spłaszcz	Option.flatten
GdyNieMaWartości	Option.defaultValue

Złożenie funkcji

```
let inline (>>) func1 func2 x = func2 (func1 x)
```

```
let liczbaZnakow str = String.length str
```

```
let razy2 x = x*2
```



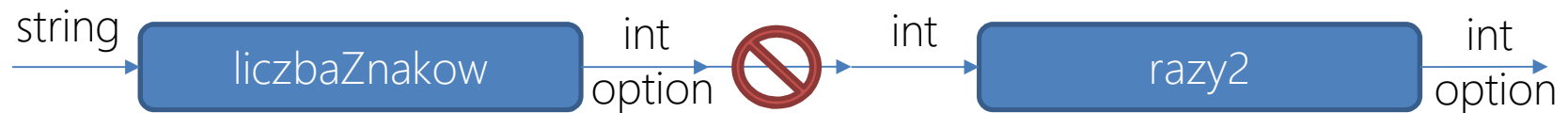
```
liczbaZnakow >> razy2
```

Złożenie funkcji monadycznych

```
let inline (>>) func1 func2 x = func2 (func1 x)
```

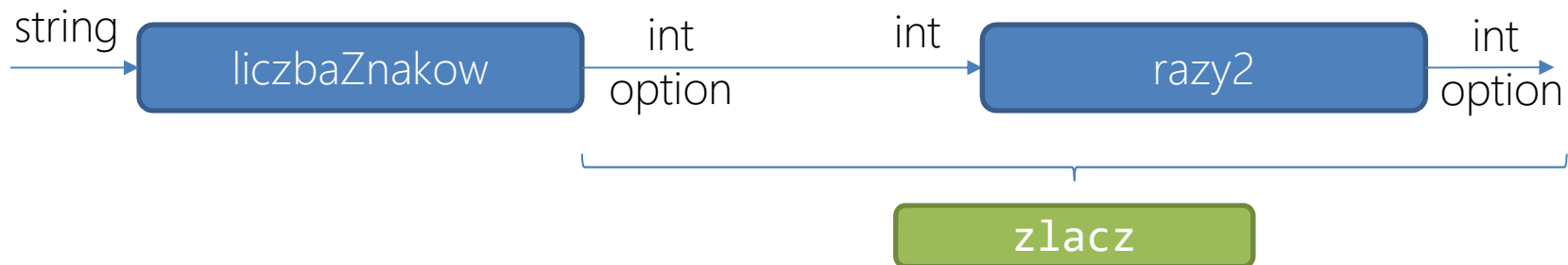
```
let liczbaZnakow str = Some(String.length str)
```

```
let razy2 x = Some(x*2)
```



Złączenie funkcji monadycznych

$\text{złącz: } M < 'T > \rightarrow ('T \rightarrow M < 'R >) \rightarrow M < 'R >$



```
let (>>=) func1 func2 x = (func1 x) |> Option.bind func2
```

```
liczbaZnakow >>= razy2
```

MONADA EITHER (RESULT)

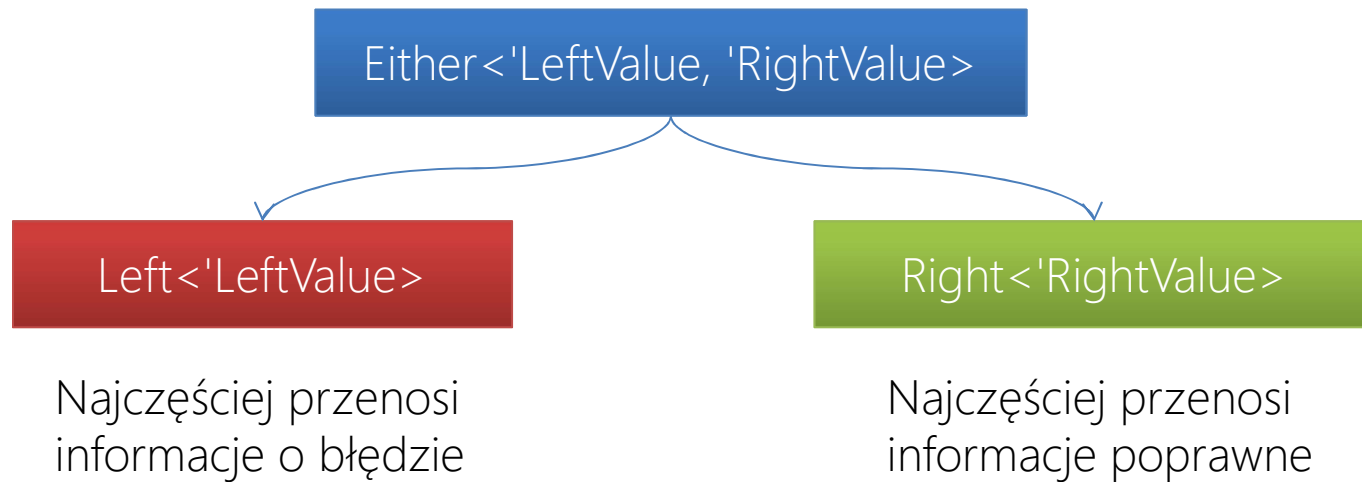
Either (Result)

Monada Either jest funkcyjną metodą pozwalającą na rozwiązanie problemu obsługi błędów

Realizacja Railway Oriented Programming

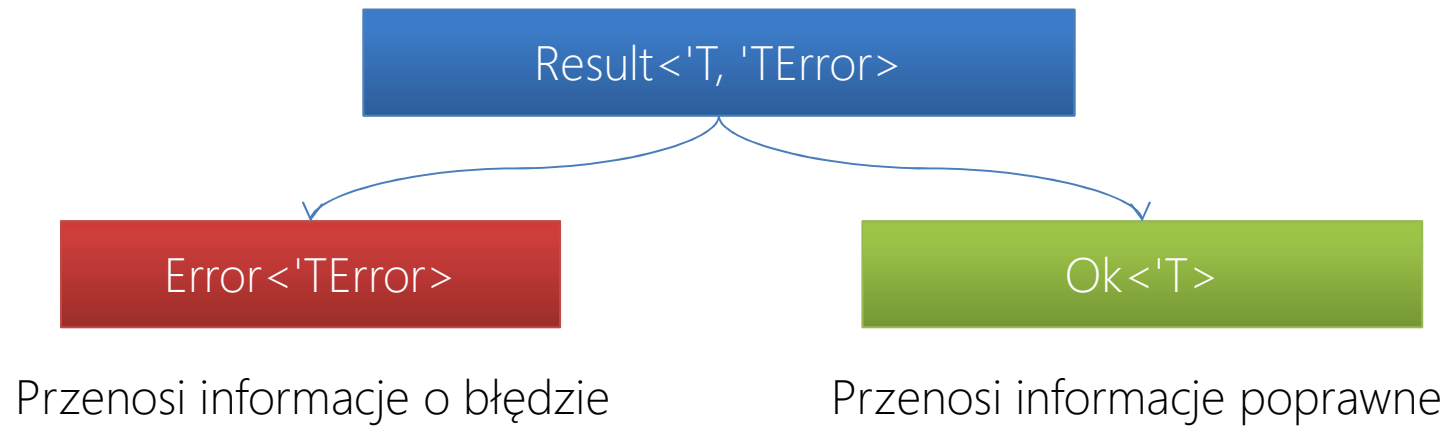


Monada Either



```
type either<'LeftValue', 'RightValue'> =  
  | Left of 'LeftValue  
  | Right of 'RightValue
```

Monada Result



```
type Result<'T, 'TError> =  
  | Ok of 'T  
  | Error of 'TError
```


Monada Result

```
let map mapaPrawa opcje =  
    match opcje with  
    | Error e -> Error e  
    | Ok x -> Ok (mapaPrawa x)
```

```
let bind mapaPrawa opcje =  
    match opcje with  
    | Error e -> Error e  
    | Ok x -> mapaPrawa x
```

Monada Result

```
let akcjaBlad akcja wynik =  
  match wynik with  
  | Error e -> akcja e  
  | Ok _ -> ()  
  wynik
```

```
let akcjaOk akcja wynik =  
  match wynik with  
  | Error _ -> ()  
  | Ok x -> akcja x  
  wynik
```

```
let akcja akcjaOk akcjaBlad =  
  function  
  | Error e -> akcjaBlad e  
  | Ok x -> akcjaOk x
```

Monada Result

```
let wczytajDane nazwaPliku =  
    try  
        File.ReadAllText nazwaPliku  
        |> JsonConvert.DeserializeObject<Osoba list>  
        |> Ok  
    with  
    | _ -> Error "Tego błędu to już nie naprawię"
```

Monada Result

```
let znajdzOsobePoId (lista:Osoba list) id =  
    let poId (osoba:Osoba) = osoba.id = id  
  
    if lista |> Seq.exists poId then  
        Ok (lista |> List.find poId)  
    else  
        Error "Nie ma osoby o takim id"
```

Monada Result

```
let parse str =  
    try  
        Ok (int str)  
    with  
        | _ -> Error "To musi być liczba"  
  
let rec loop f =  
    match f () with  
    | Ok x -> x  
    | Error e -> Console.Clear()  
                pokazBład e  
                loop f
```

Monada Result

```
let przetworz dane =  
    loop wczytajId  
    |> znajdzOsobePoId dane  
    |> Result.map naString
```

```
[<EntryPoint>]
```

```
let main argv =  
    wczytajDane "dane.json"  
    |> Result.bind przetworz  
    |> akcja (printfn "%s") pokazBlad
```

Co powinna zwrócić ta funkcja?

Czy poniższa funkcja powinna zwracać wartość typu Result?

```
let znajdzOsobePoId (lista:Osoba list) id =  
    let poId (osoba:Osoba) = osoba.id = id  
  
    if lista |> Seq.exists poId then  
        Ok (lista |> List.find poId)  
    else  
        Error "Nie ma osoby o takim id"
```

Co powinna zwrócić ta funkcja?

A jeżeli będzie zwracała opcję?

```
let znajdzOsobePoId (lista:Osoba list) id =  
    let poId (osoba:Osoba) = osoba.id = id  
  
    if lista |> Seq.exists poId then  
        Some (lista |> List.find poId)  
    else  
        None
```


Jak to zmieni funkcję main?

```
[<EntryPoint>]
```

```
let main argv =
```

```
    wczytajDane "dane.json"
```

```
    |> Result.map (pair (loop wczytajId))
```

```
    |> Result.map (fun (d, id) -> znajdzOsobePoId d id)
```

```
    |> Result.map (Option.map naString)
```

```
    |> Result.map  
        (Option.defaultValue "Nie ma osoby o takim id")
```

```
    |> action (printfn "%s") pokazBlad
```

Osoba list->int->Osoba option

Result<Osoba option, string>