

WZORCE PROGRAMOWANIA FUNKCYJNEGO

Aplikacja funkcji

Aplikacja funkcji jest procesem wyznaczenia wartości funkcji dla określonych argumentów

```
let ink a = a + 1                                //ink 1 -> 2
```

`ink: int -> int`

```
let dodaj a b = a+b                             //dodaj 1 2 -> 3
```

`dodaj: int -> int -> int`

Aplikacja częściowa

```
let dodaj a b = a+b;;
```

```
let dodaj2 = dodaj 2;;
```



Powstaje nowa funkcja
jednego parametru

```
let wynik = dodaj2 4;;
```

Aplikacja częściowa

Aplikacja częściowa jest to proces aplikacji funkcji tylko do części parametrów

```
let f3 a x v = if x%v = 0 then a+x else a
```

```
let g v = fun a x -> if x%v = 0 then a+x else a;;
```

```
List.fold (g 3) 0 [1;2;3;4;5]
```

Ten argument musi mieć typ: ('a->'b->'a)

Aplikacja częściowa

Aplikacja częściowa jest to proces aplikacji funkcji tylko do części parametrów

```
let f3 v a x = if x%v = 0 then a+x else a
```

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

```
List.fold (f3 3) 0 [1;2;3;4;5]
```

Currying

Umożliwia traktowanie funkcji wielu parametrów
jako funkcji jednego parametru

```
let dodaj a b = a+b;;
```

```
val dodaj : a:int -> b:int -> int
```

Operator pipe |>

1. Dana lista

2. Odfiltrować elementy mniejsze niż 3

3. Pozostałe podnieść do kwadratu

4. Z wyników zbudować kolejną listę

```
let res = List.map square  
            (List.filter (fun x->x>3) [1;2;3;4;5;6;7] )
```

Operator pipe |>

```
let res = List.map square  
          (List.filter (fun x->x>3) [1;2;3;4;5;6;7])
```

Operator pipe pozwala na odwrócenie kolejności:
najpierw piszemy dane, a później funkcję

```
let (|>) x f = f x
```


Operator pipe |>

```
let res = List.map square  
          (List.filter (fun x->x>3) [1;2;3;4;5;6;7])
```

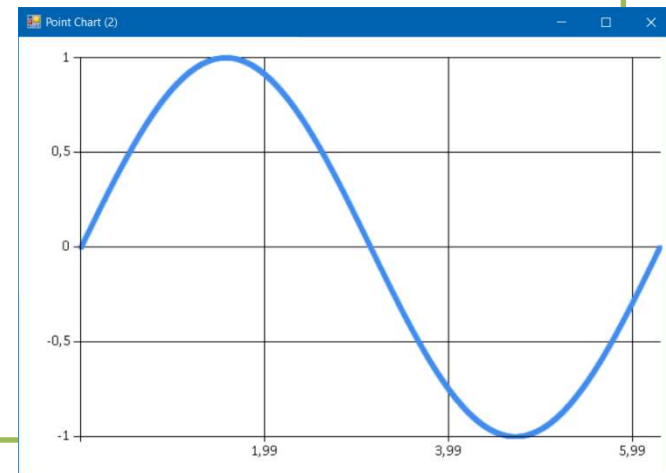
```
let res = [1;2;3;4;5;6;7]  
          |> List.filter (fun x->x>3)  
          |> List.map square
```

Operator pipe |>

```
#load  
"../packages/FSharp.Charting.0.90.14/FSharp.Charting.fsx"
```

```
open System  
open FSharp.Charting
```

```
[0.0..0.01..2.0*Math.PI]  
|> List.map (fun x -> (x, sin x))  
|> Chart.Point
```



Operator pipe |>

```
let rec agregacja pocz funkcja = function  
| Pusta -> pocz  
| Element (g,o) -> funkcja (agregacja pocz funkcja o) g
```

```
lista |> agregacja 0 (podzielnePrzez 3)
```

Inne pipe

```
(1,2) ||> min
```

```
(1,2,3) ||> (fun a b c -> a+b+c)
```

```
(fun a -> a+1) <| 1
```

```
min <| (1,2)
```

```
(fun a b c -> a+b+c) <| (1,2,3)
```

Złożenie funkcji

```
let res = [1;2;3;4;5;6;7]  
          |>List.filter (fun x->x>3)  
          |>List.map square
```

```
let f lista = lista  
            |> List.filter (fun x->x>3)  
            |> List.map square
```

Złożenie funkcji

$$f \circ g = g(f(x))$$

```
let square x = x*x  
let double x = x+x
```

```
let doubleSquare = square >> double
```



```
let doubleSquare x = double (square x)
```

Złożenie funkcji

```
let f lista = lista  
    |> List.filter (fun x->x>3)  
    |> List.map square
```

```
let f = List.filter (fun x->x>3)  
    >> List.map square
```

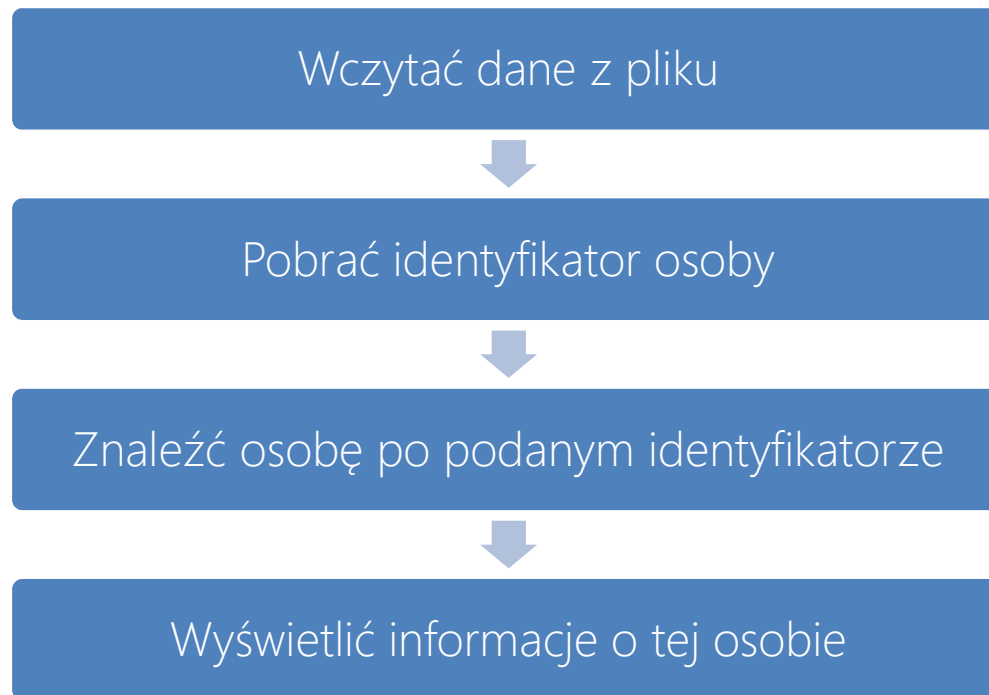
Złożenie funkcji

```
let f = List.filter (fun x->x>3)  
    >> List.map square
```



```
let log x = printfn "%A" x  
            x  
let fWithLog = log>>f>>log
```


Przykładowa aplikacja



Przykładowa aplikacja

Dane dla naszego przykładu

```
[<AllowNullLiteral>]  
type Osoba(id:int, imie:string, nazwisko:string) =  
    member this.id = id  
    member this.imie = imie  
    member this.nazwisko = nazwisko
```

Przykładowa aplikacja

```
let wczytajDane nazwaPliku =  
    File.ReadAllText nazwaPliku  
    |> JsonConvert.DeserializeObject<Osoba list>
```

```
let wyswietlDane (osoba:Osoba) =  
    printfn "%d" osoba.id  
    printfn "%s" osoba.imie  
    printfn "%s" osoba.nazwisko
```

```
let znajdzOsobePoId (lista:Osoba list) id =  
    let poId (osoba:Osoba) = osoba.id = id  
    if lista |> Seq.exists poId then  
        lista |> List.find poId  
    else  
        null
```

Przykładowa aplikacja

```
[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane.json"
    printf "Podaj id osoby: "
    Console.ReadLine ()
    |> int
    |> znajdzOsobePoId dane
    |> wyswietlDane
```

Co w tej aplikacji może pójść źle?



```
[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane.json"
    printfn "Podaj id osoby: "
    Console.ReadLine ()
    |> int
    |> znajdzOsobePoId dane
    |> wyswietlDane
```

Co w tej aplikacji może pójść źle?

```
let main argv =  
    let dane = wczytajDane "dane.json"  
    printf "Podaj id osoby: "  
    Console.ReadLine ()  
    |> int  
    |> znaj  
    |> wysw
```



Może rzucić 9 różnych typów wyjątków

Może rzucić wyjątek `JsonReaderException`

```
let wczytajDane nazwaPliku =  
    File.ReadAllText nazwaPliku  
    |> JsonConvert.DeserializeObject<Osoba list>
```


Co w tej aplikacji może pójść źle?

```
let main argv =  
    let dane = wczytajDane "dane.json"  
    printf "Podaj id osoby: "  
    Console.ReadLine ()  
    |> int  
    |> znajdzOsobePoId dane  
    |> wyswietlDane
```

Mogą być rzucone wyjątki, ale są mało prawdopodobne

Co w tej aplikacji może pójść źle?

```
let main argv =  
    let dane = wczytajDane "dane.json"  
    printf "Podaj id osoby:"  
    Console.ReadLine ()  
    |> int  
    |> znajdzOsobePoId dane  
    |> wyswietlDane
```



String może być w złym formacie

Co w tej aplikacji może pójść źle?

```
let main argv =  
    let dane = wczytajDane "dane.json"  
    printf "Podaj id osoby:"  
    Console.ReadLine ()  
    |> int  
    |> znajdzOsobePoId dane  
    |> wyswietlDane
```

Ten fragment jest
bezpieczny

Co w tej aplikacji może pójść źle?

```
let main argv =  
    let dane = wczytajDane "dane.json"  
    printf "Podaj id osoby:"  
    Console.ReadLine ()  
    |> int  
    |> znajdzOsobePoId dane  
    |> wyswietlDane
```

```
let wyswietlDane (osoba:Osoba) =  
    printfn "%d" osoba.id  
    printfn "%s" osoba.imie  
    printfn "%s" osoba.nazwisko
```



Jeżeli osoba będzie
null to będzie wyjątek

Obsługa wyjątków



Naprawa aplikacji

```
[<EntryPoint>]
let main argv =
    try
        let dane  = wczytajDane "dane.json"
        let id    = wczytajId ()
        let osoba = znajdzOsobePoId dane id
        if osoba <> null then
            wyswietlDane osoba
        else
            pokazBlad "Nie ma osoby o takim id"
    with
    | _ -> pokazBlad "Tego błędu już nie naprawię"
0
```

Pomyłka za miliardy dolarów



Tony Hoare

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language ([ALGOL W](#)). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

Naprawa aplikacji

```
let rec wczytajId () =  
    try  
        printf "Podaj id osoby: "  
        Console.ReadLine () |> int  
    with  
        | _ -> Console.Clear();  
            pokazBład "Błędny identyfikator użytkownika"  
            wczytajId ()
```

Naprawa aplikacji

```
let pokazBlad (wiadomosc:string) =  
    Console.ForegroundColor <- ConsoleColor.Red  
    printfn "%s" wiadomosc  
    Console.ResetColor();
```

```

let rec wczytajId () =
    try
        printf "Podaj id osoby: "
        Console.ReadLine () |> int
    with
        | _ -> Console.Clear();
            pokazBład "Błędny identyfikator użytkownika"
            wczytajId ()

[<EntryPoint>]
let main argv =
    try
        let dane = wczytajDane "dane.json"
        let id = wczytajId ()
        let osoba = znajdzOsobePoId dane id
        if osoba <> null then
            wyswietlDane osoba
        else
            pokazBład "Nie ma osoby o takim id"
    with
        | _ -> pokazBład "Tego błędu już nie naprawię"
    0

```

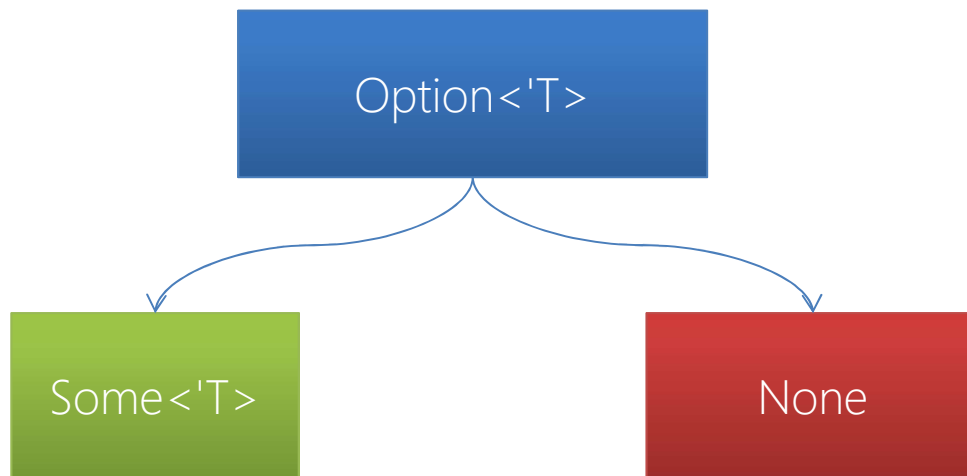


Czy można to napisać lepiej?

BO JAK NIE NULL TO CO?

Opcje

Typ Option pozwala na określanie czy dana wartość istnieje czy nie



```
type Option<'T> =  
| Some of 'T  
| None
```

```
let znajdzOsobePoId (lista:Osoba list) id =  
    let poId (osoba:Osoba) = osoba.id = id  
    if lista |> Seq.exists poId then  
        lista |> List.find poId  
    else  
        null
```

Osoba list->int->Osoba



```
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
```

Osoba list->int->Option<Osoba>

```
let dane = wczytajDane "dane.json"
let id    = wczytajId ()
let osoba = znajdzOsobePoId dane id
if osoba <> null then
    wyswietlDane osoba
else
    pokazBlad "Nie ma osoby o takim id"
```



```
let dane = wczytajDane "dane.json"
let id    = wczytajId ()
let osoba = znajdzOsobePoId dane id
match osoba with
| Some osoba -> wyswietlDane osoba
| None -> pokazBlad "Nie ma osoby o takim id"
```

```
let pokazWynik = function
  | Some osoba -> wyswietlDane osoba
  | None -> pokazBlad "Nie ma osoby o takim id"
```

```
let dane = wczytajDane "dane.json"
let id = wczytajId ()
let osoba = znajdzOsobePoId dane id
pokazWynik osoba
```



```
let dane = wczytajDane "dane.json"
wczytajId ()
|> znajdzOsobePoId dane
|> pokazWynik
```

funkcja main

```
[<EntryPoint>]
let main argv =
    try
        let dane = wczytajDane "dane.json"
        wczytajId ()
        |> znajdzOsobePoId dane
        |> pokazWynik
    with
    | _ -> pokazBlad "Tego błędu już nie naprawię"
    0
```

funkcja wczytajId

```
let rec wczytajId () =  
    try  
        printfn "Podaj id osoby: "  
        Console.ReadLine () |> int  
    with  
    | _ -> Console.Clear();  
        pokazBład "Błędny identyfikator użytkownika"  
        wczytajId ()
```

funkcja wczytajId

```
let rec wczytajId () =  
    try  
        printfn "Podaj id osoby: "  
        Console.ReadLine () |> int  
    with  
    | _ -> Console.Clear();  
        pokazBlad "..."  
        wczytajId ()
```



```
let wczytajId () =  
    printfn "Podaj id osoby: "  
    Console.ReadLine () |> parse
```

```
let parse str =  
    try  
        Some (int str)  
    with  
    | _ -> None
```


Opcje w tworzeniu pętli

```
let rec loop f =  
  match f () with  
  | Some x -> x  
  | None -> loop f
```

```
[<EntryPoint>]  
let main argv =  
  let dane = wczytajDane "dane.json"  
  
  loop wczytajId  
  
  |> znajdzOsobePoId2 dane  
  |> pokazWynik  
  0
```

Funkcja wczytajDane

```
let wczytajDane nazwaPliku =  
    try  
        File.ReadAllText nazwaPliku  
        |> JsonConvert.DeserializeObject<Osoba list>  
        |> Some  
    with  
    | _ -> pokazBlad ("Tego błędu to już nie naprawię")  
        None
```

```
[<EntryPoint>]
```

```
let main argv =
```

```
    let dane = wczytajDane "dane.json"
```

```
    loop wczytajId
```

```
    |> znajdzOsobePoId dane
```

```
    |> pokazWynik
```

```
0
```

string → Osoba list option



Nie
pasuje

Osoba list → int -> Osoba option

```
[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane.json"
    match dane with
    | Some osoby ->
        loop wczytajId
        |> znajdzOsobePoId osoby
        |> pokazWynik
    | None -> ()
```

0

Komendy/akcje i wyrażenia

Komendy / akcje

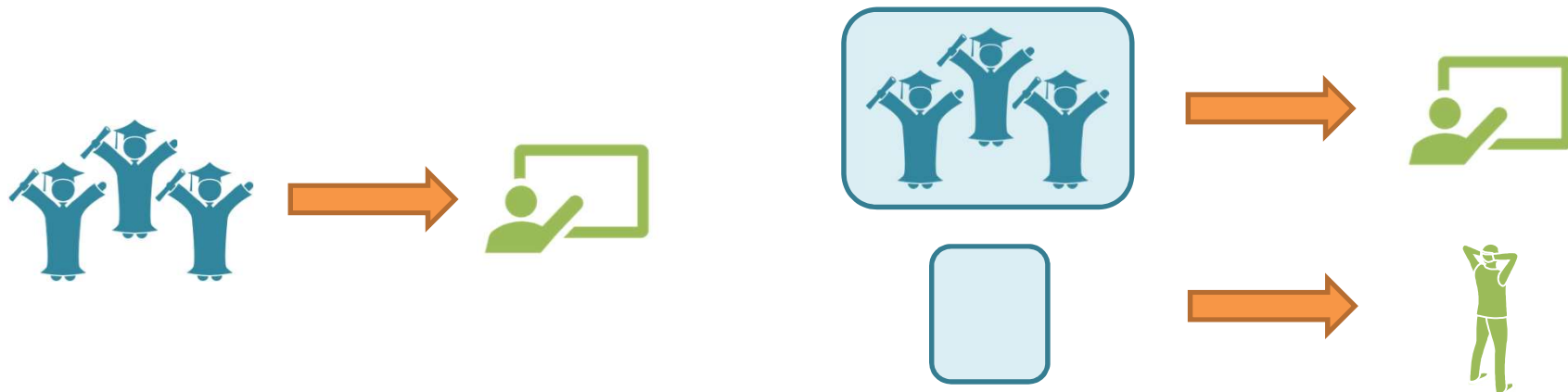
- Modyfikują stan programu
- Ich działanie można obserwować tylko poprzez analizę stanu

Wyrażenia

- Tworzą wartości
- Ich rezultat można obserwować poprzez analizę wartości zwracanej

Funkcja wykonaj

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



Funkcja wykonaj

```
[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane.json"
    wykonaj (
        fun dane->
            loop wczytajId
            |> znajdzOsobePoId dane
            |> pokazWynik
    ) dane
    0
```

Funkcja main

```
[<EntryPoint>]
let main argv =
    wczytajDane "dane.json"
    |> wykonaj (fun dane->
        loop wczytajId
        |> znajdzOsobePoId dane
        |> pokazWynik
    )
0
```

Operacje wejściowe



Rdzeń aplikacji (czysty)



Operacje wyjściowe

To co byśmy chcieli uzyskać

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