

Paradygmaty programowania

WPROWADZENIE

dr inż. Łukasz Bartczuk

Instytut Inteligentnych Systemów Informatycznych

p. 517

Lukasz.Bartczuk@iisi.pcz.pl

iisi.pcz.pl/

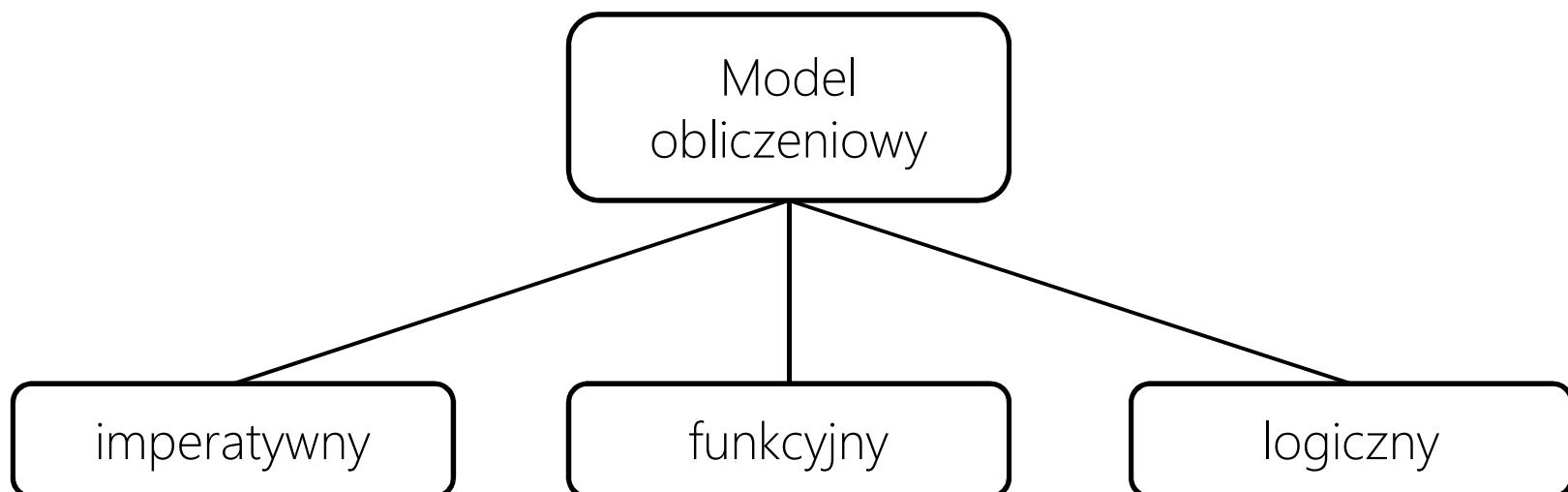
Konsultacje: wtorek 12-14, środa 9-11

Co to jest komputer?

Co to jest
program komputerowy?

Model obliczeniowy

Model obliczeniowy jest to zbiór wartości (prostych lub złożonych), skojarzonych z nimi operacji oraz operacji wykorzystywanych do zdefiniowania obliczeń.



Model funkcyjny

Model ten składa się z wartości i funkcji (przy czym funkcje też są wartością), a podstawową operacją do wykonywania obliczeń jest aplikacja funkcji do wartości.

$$f(x) = x^2 + x + 5$$

$$f(2) \rightarrow 2^2 + 2 + 5 \rightarrow 11$$

$$f(\sin(y)) \rightarrow \sin(y)^2 + \sin(y) + 5$$

$$\rightarrow g(y) = \sin(y)^2 + \sin(y) + 5$$

Model logiczny

Model ten składa się z faktów, relacji i zapytań, a metodą dokonywania obliczeń jest wnioskowanie logiczne.

1. student(tomek).
2. LubiParadygmaty(X) jeżeli student(X).
3. ?- LubiParadygmaty(Y).

4. student(Y).

5. Y = tomek.

Model imperatywny

Model imperatywny składa się wartości oraz stanu i operacji przypisania.

$$\{x=2, y=3\}$$

$$y := x^2 + x + 5$$

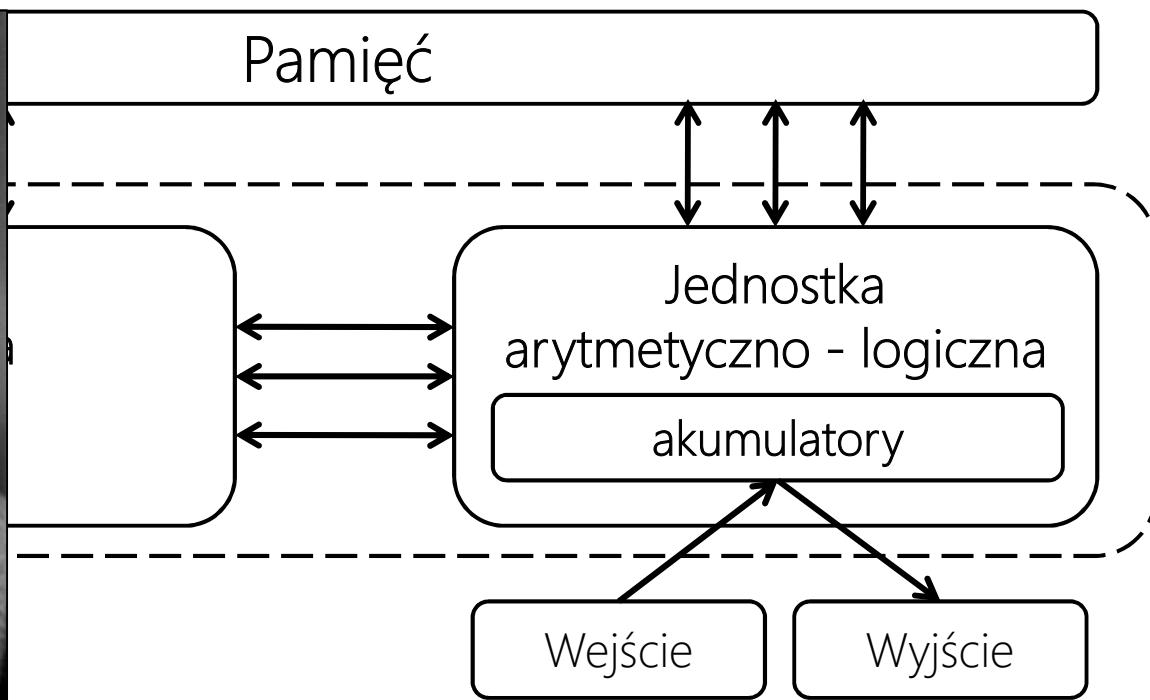
$$\{x=2, y=11\}$$

$$\{x=2, y=11\}$$

$$y := y + 5$$

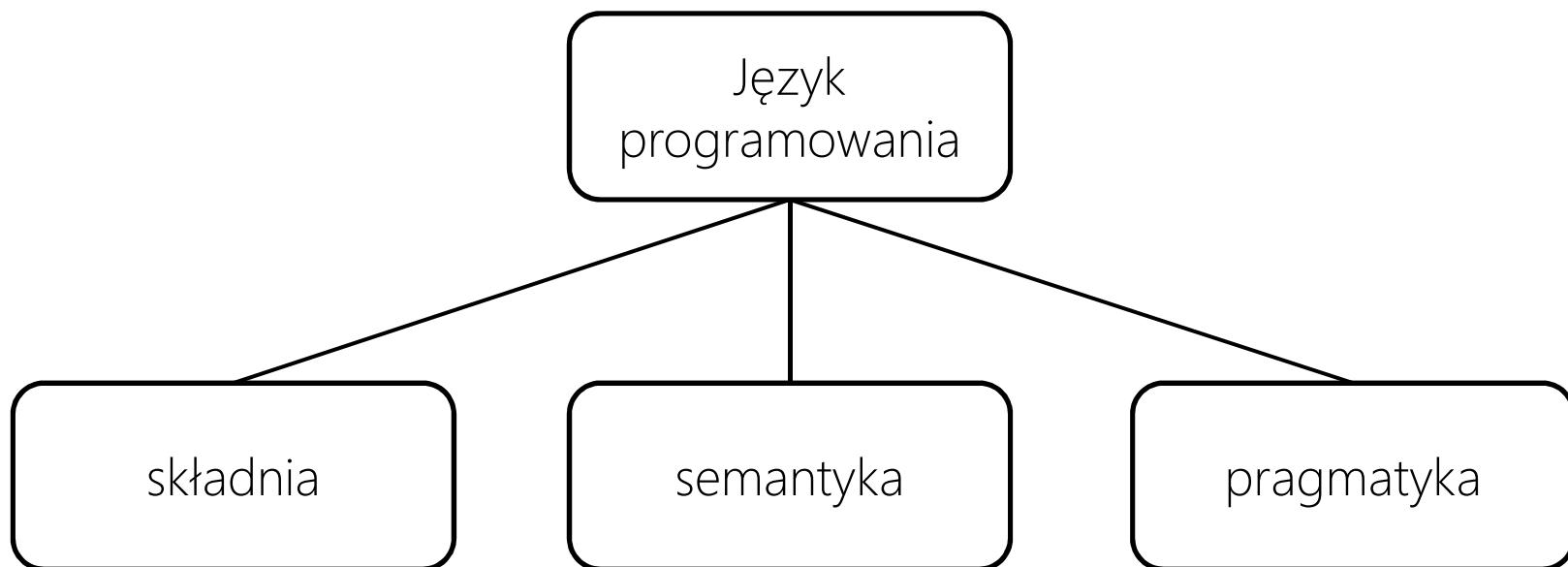
$$\{x=2, y=16\}$$

Maszyna John'a von Neumanna



1903-1957

Co to jest język programowania?



Składnia języków programowania

Zbiór zasad określających, który ciąg znaków jest poprawną instrukcją w danym języku programowania.

Do definiowania składni języka programowania najczęściej używa się notacji BNF (Backus-Naur Form)

```
<expression> ::= <term> |
                  <expression> '+' <term> |
                  <expression> '-' <term>
<term>      ::= <factor> |
                  <term> '*' <factor> |
                  <term> '/' <factor>
<factor> ::= <number> | '(' <expression> ')' 
```

Semantyka języków programowania

Określa relację pomiędzy elementami składni języka programowania, a modelem obliczeniowym.

Semantyka aksjomatyczna

Semantyka denotacyjna

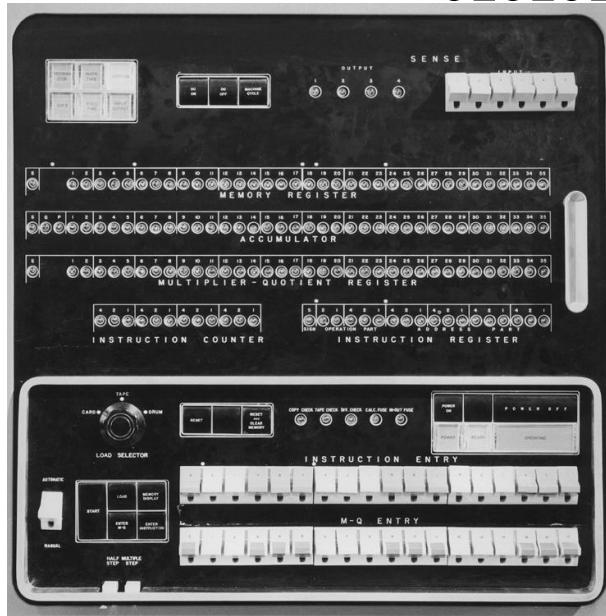
Semantyka operacyjna

Pragmatyka języków programowania

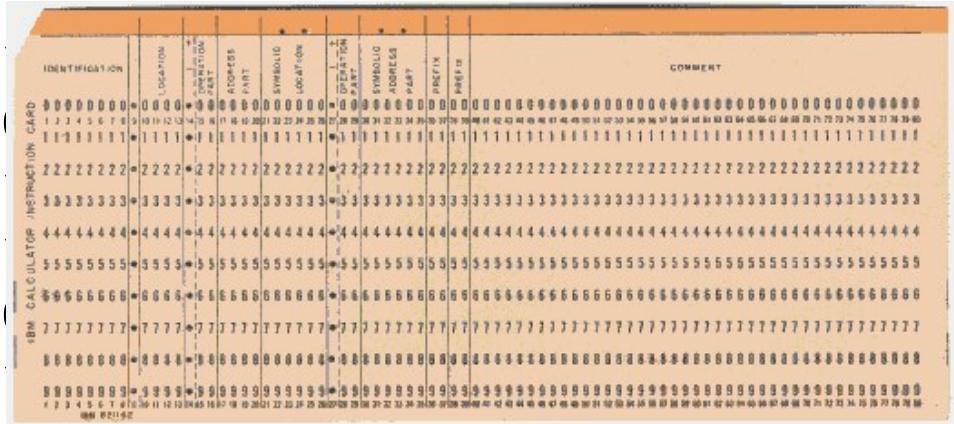
Opisuje praktyczne aspekty implementacji języka i korzystania z niego. Określa m.in. jak konstrukcje i cechy języka mogą być wykorzystane do osiągnięcia określonego celu.

Ewolucja języków programowania

Początkowo programy wyglądały tak:



```
010101011000100  
000001000  
000000010  
001100011  
100000000  
100100010  
01111101001000110100000100  
1011110100010001011111100  
0000000000000000000000000000000  
0011001001110000111001000
```



Ewolucja języków programowania

Funkcja w języku assembler
(nowoczesnym)



1919 - 2001

```
push ebp
mov ebp, esp
sub esp, 0x10
mov DWORD PTR [ebp-0xc], 0x5
mov DWORD PTR [ebp-0x8], 0x6
mov eax, DWORD PTR [ebp-0x8]
mov edx, DWORD PTR [ebp-0xc]
lea eax, [edx+eax*1]
mov DWORD PTR [ebp-0x4], eax
mov eax, 0x0
leave
ret
```

Ewolucja języków programowania

... i w C

```
int main()
{
    int a = 5;
    int b = 6;
    int c = a+b;
    return 0;
}
```

Ewolucja języków programowania

Twórca pierwszego
kompilatora

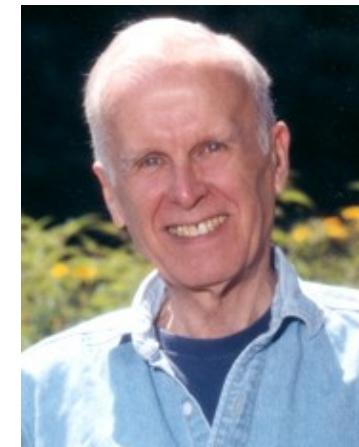
kadm. Grace Hooper



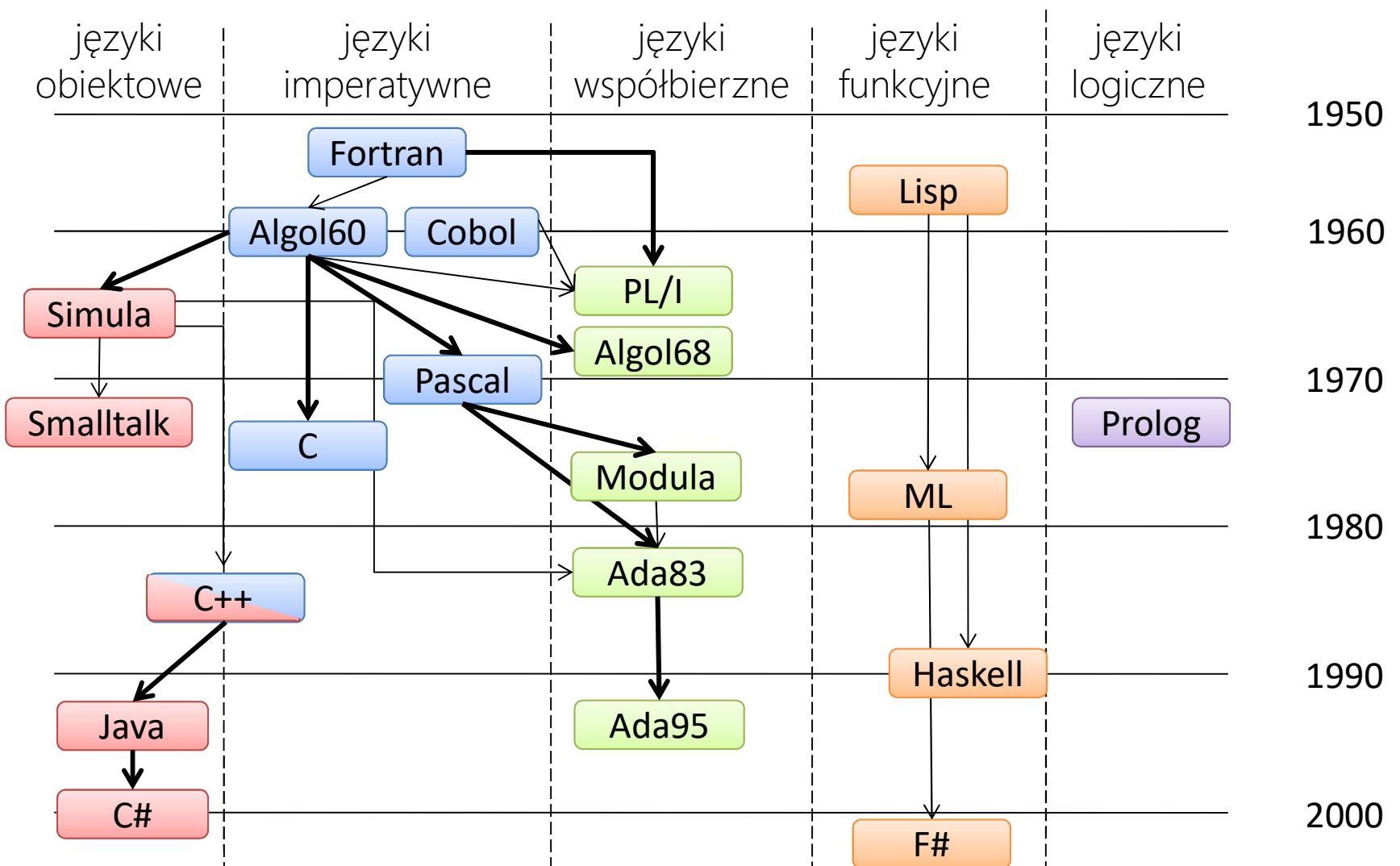
1906 - 1992

Twórca języka
Fortran

John Backus



1924 - 2007

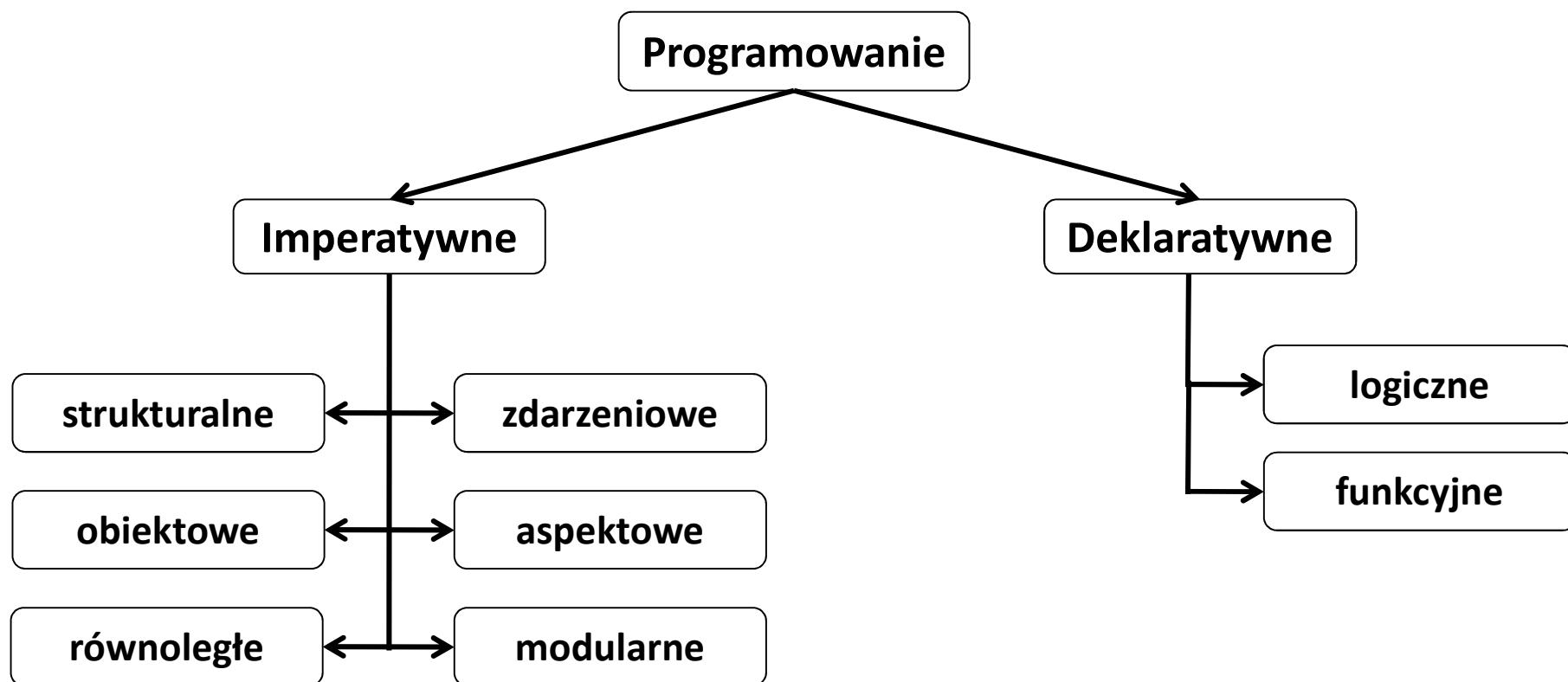


Paradygmat programowania

Paradygmat programowania jest wyróżniającym się stylem programowania. Każdy paradygmat jest scharakteryzowany przez dominację pewnych kluczowych koncepcji.

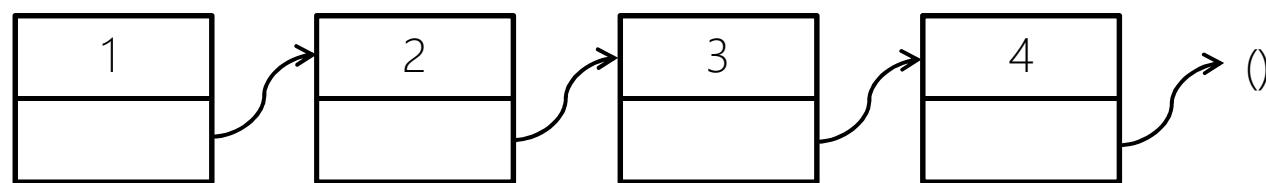
- zbiór koncepcji reprezentujących podejście do implementacji algorytmów
- zbiór mechanizmów używanych przez programistę do pisania programów i określających jak te programy będą następnie wykonywane przez komputer

Paradygmat programowania



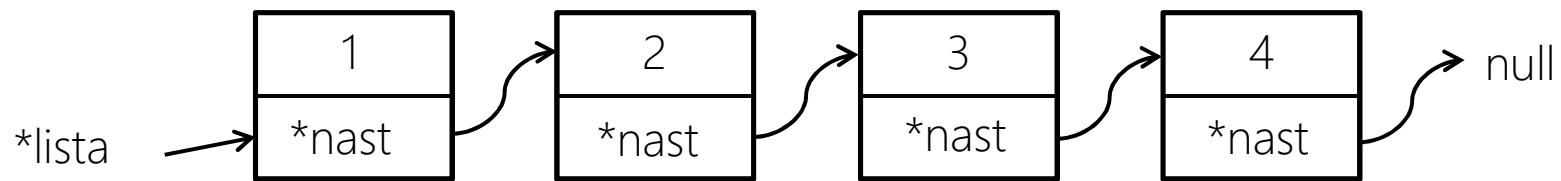
Problem do rozwiązania

Dana jest lista liczb całkowitych:



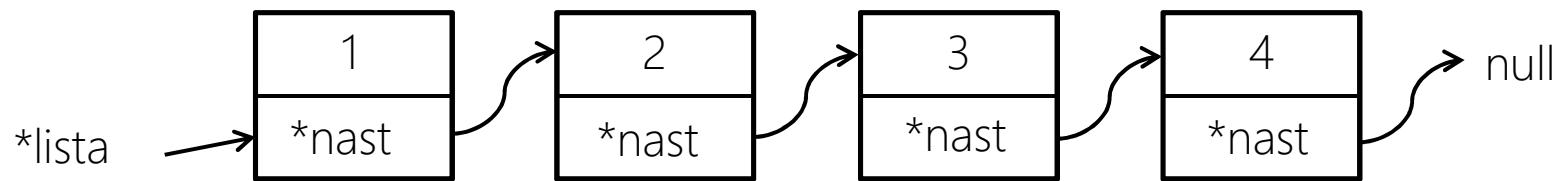
Napisać funkcję, która będzie obliczała sumę elementów na liście

Programowanie imperatywne



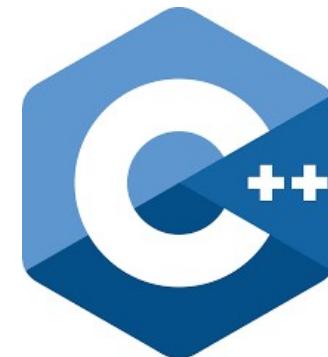
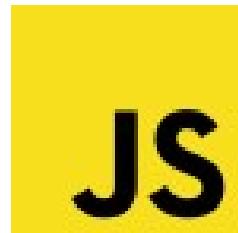
1. Stwórz zmienną **biezacy** określającą aktualny element listy i przypisz do niej adres pierwszego elementu na liście (parametr funkcji)
2. Stwórz zmienną **suma** przechowującą sumę wartości kolejnych elementów listy
3. W pętli (dopóki zmienienna **biezacy** ma inną wartość niż **null**):
 1. Pobierz wartość składowej **wartosc** aktualnego elementu listy, pobierz wartość zmiennej **suma** zawierającą sumę wszystkich dotychczasowych elementów. Dodaj te dwie wartości do siebie. Wynik przypisz do zmiennej **suma**.
 2. Pobierz wartość wskaźnika **nast** aktualnego elementu na liście i przypisz go do zmiennej **biezacy**
4. Zwróć wartość zmiennej **suma** do programu wywołującego

Programowanie imperatywne

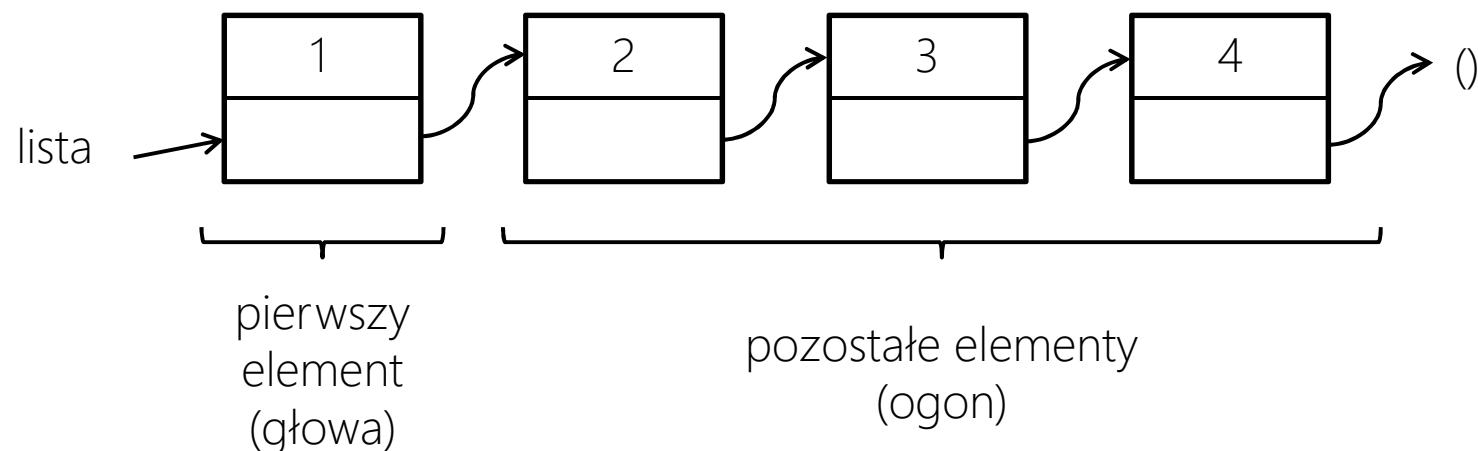


```
int Sumuj(Lista* lista) {
    Lista* biezacy = lista;
    int suma = 0;
    while(biezacy != NULL) {
        suma = suma + biezacy->wartosc;
        biezacy = biezacy->nast;
    }
    return suma;
}
```

Programowanie imperatywne

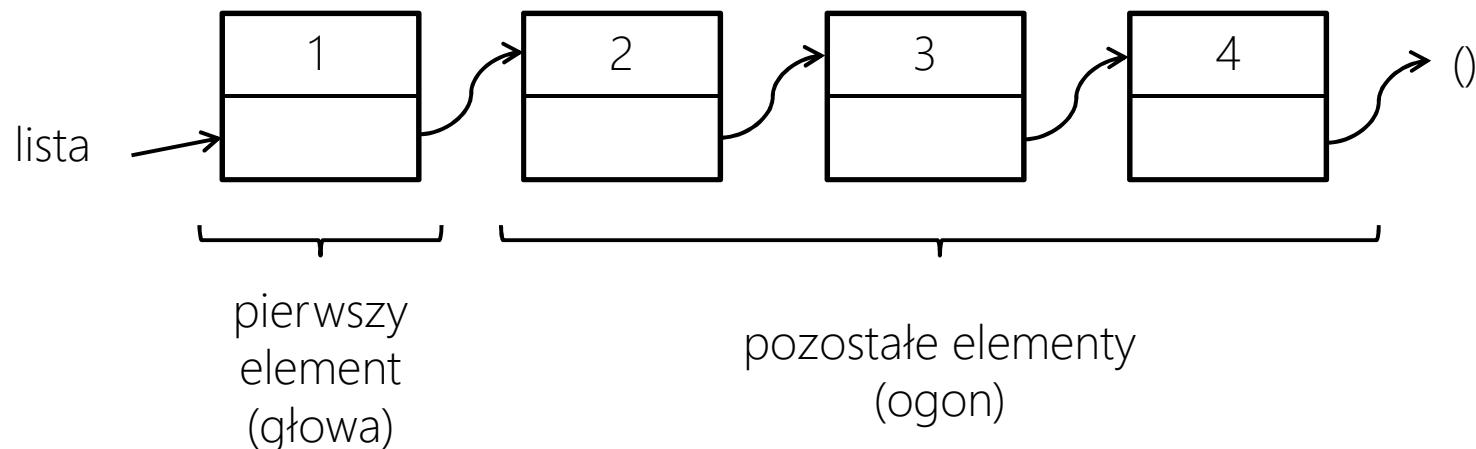


Programowanie funkcyjne



1. Jeżeli lista jest pusta to zwróć 0.
2. Jeżeli lista nie jest pusta to pobierz wartość pierwszego elementu na liście i dodaj go do sumy pozostałych elementów na liście

Programowanie funkcyjne



```
let rec sumuj lista =
    if List.isEmpty lista then
        0
    else
        lista.Head + sumuj lista.Tail;;
```

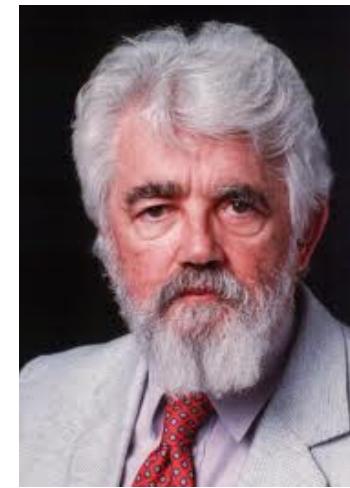
Programowanie funkcyjne

Alonzo Church
(twórca rachunku lambda)



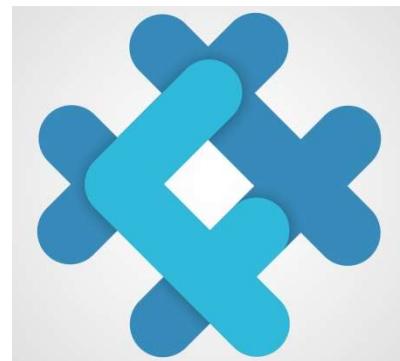
1903 - 1995

John McCarthy
(projektant języka Lisp)

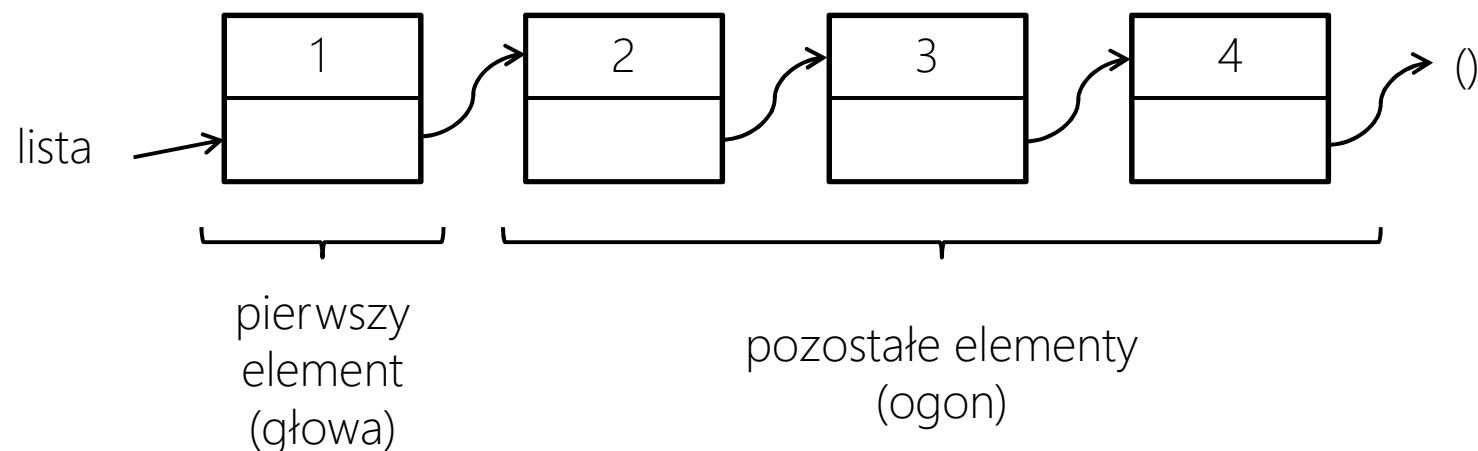


1927 - 2011

Programowanie funkcyjne

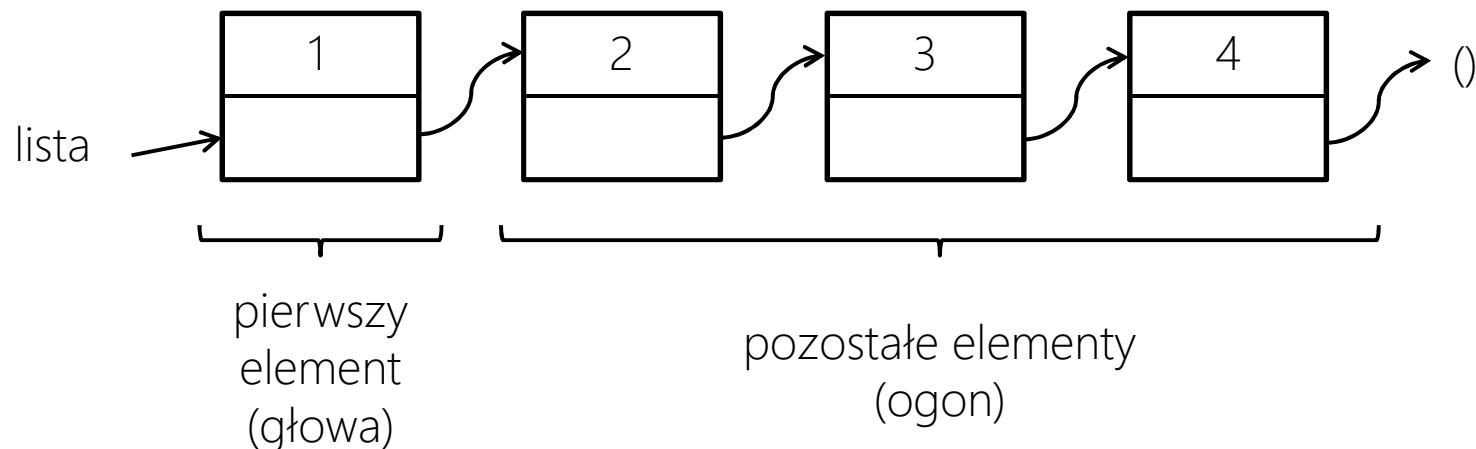


Programowanie logiczne



1. Jeżeli lista jest listą pustą to prawdą jest, że suma jej elementów Y równa się 0.
2. Jeżeli lista ma głowę X i ogon Xs oraz suma elementów w ogonie to Suma, to prawdą jest, że odpowiedzią jest X plus Suma

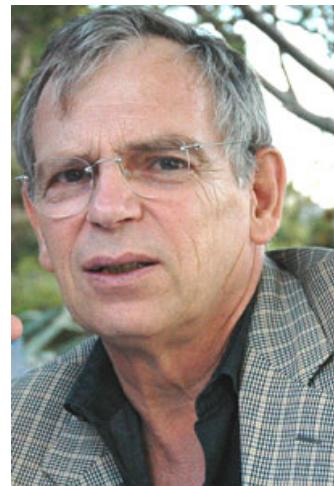
Programowanie logiczne



```
sumuj([], Y) :- Y is 0.  
sumuj([X|Xs], Y) :- sumuj(Xs, Suma), Y is Suma+X.
```

Programowanie logiczne

Alain Colmerauer
Twórca języka Prolog



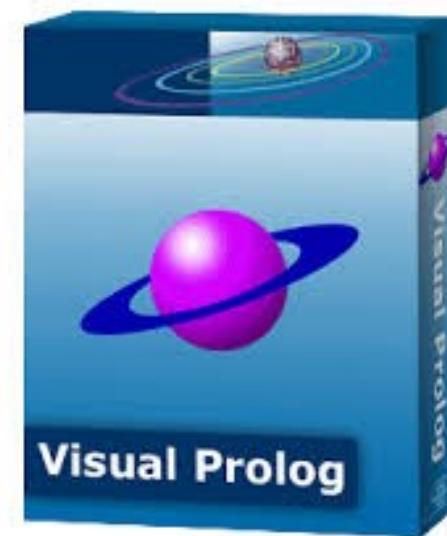
1941-

Robert "Bob" Anthony Kowalski
Twórca proceduralnej interpretacji
klauzul Horna



1941-

Programowanie logiczne

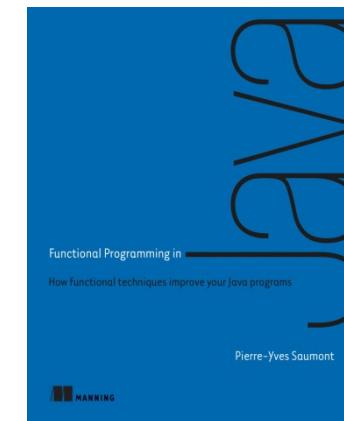
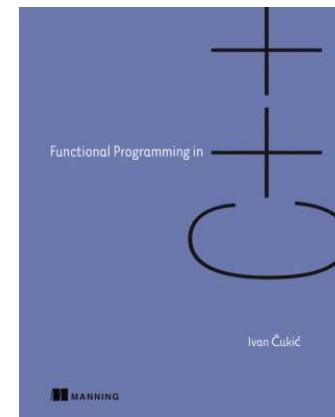
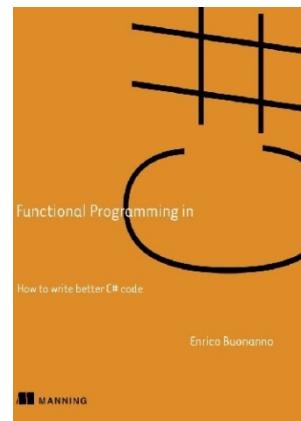
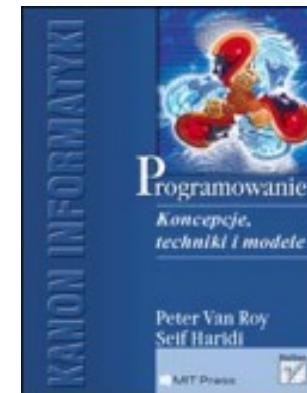
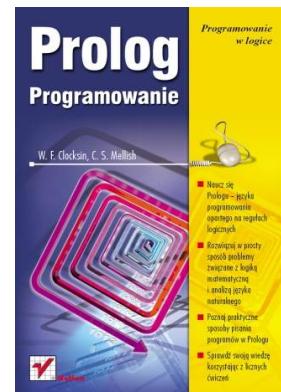
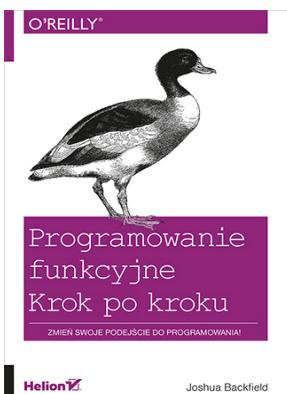


Zaliczenie przedmiotu

Przedmiot kończy się
egzaminem

1. Do egzaminu będą dopuszczone tylko osoby, które uzyskały zaliczenie laboratorium.
2. Nie będzie zwolnień z egzaminu.
3. Egzamin będzie przeprowadzony w formie testowej.

Literatura

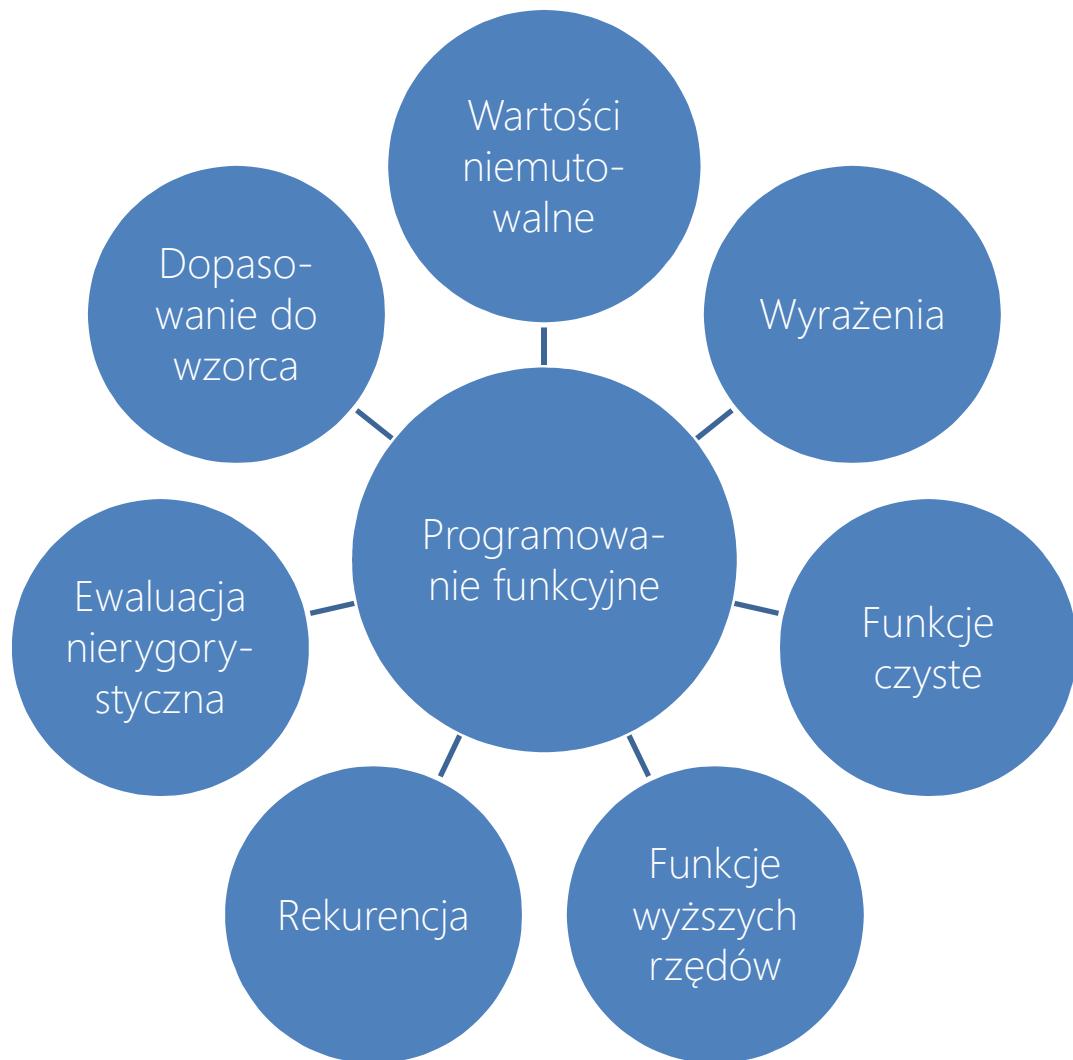


Materiały w internecie

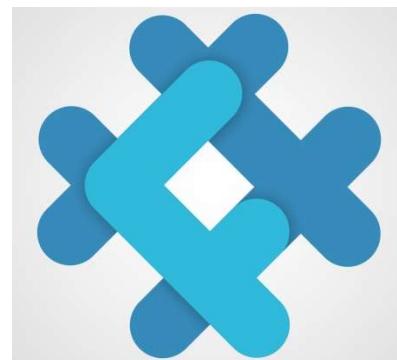
- Coursera
- Pluralsight

PROGRAMOWANIE FUNKCYJNE

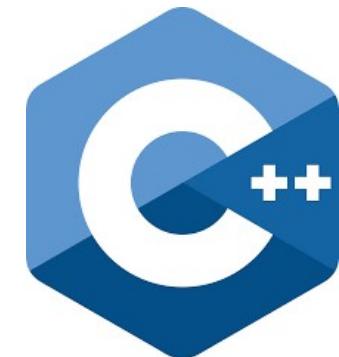
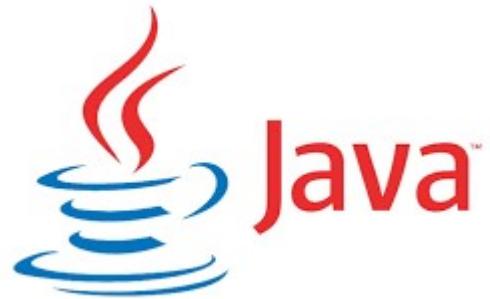
Programowanie funkcyjne jest
to programowanie
z funkcjami matematycznymi.



Programowanie funkcyjne



Języki z cechami funkcyjnymi



F# |> I ❤

JĘZYK F#

Język F#

F# jest nowoczesnym językiem programowania typu first - functional

F# ma deklaratywną składnię, która pozwala na skupienie się na tym „co”, a nie „jak” zrobić

```
let square x = x*x  
let squared = List.map square [1;3;5]
```



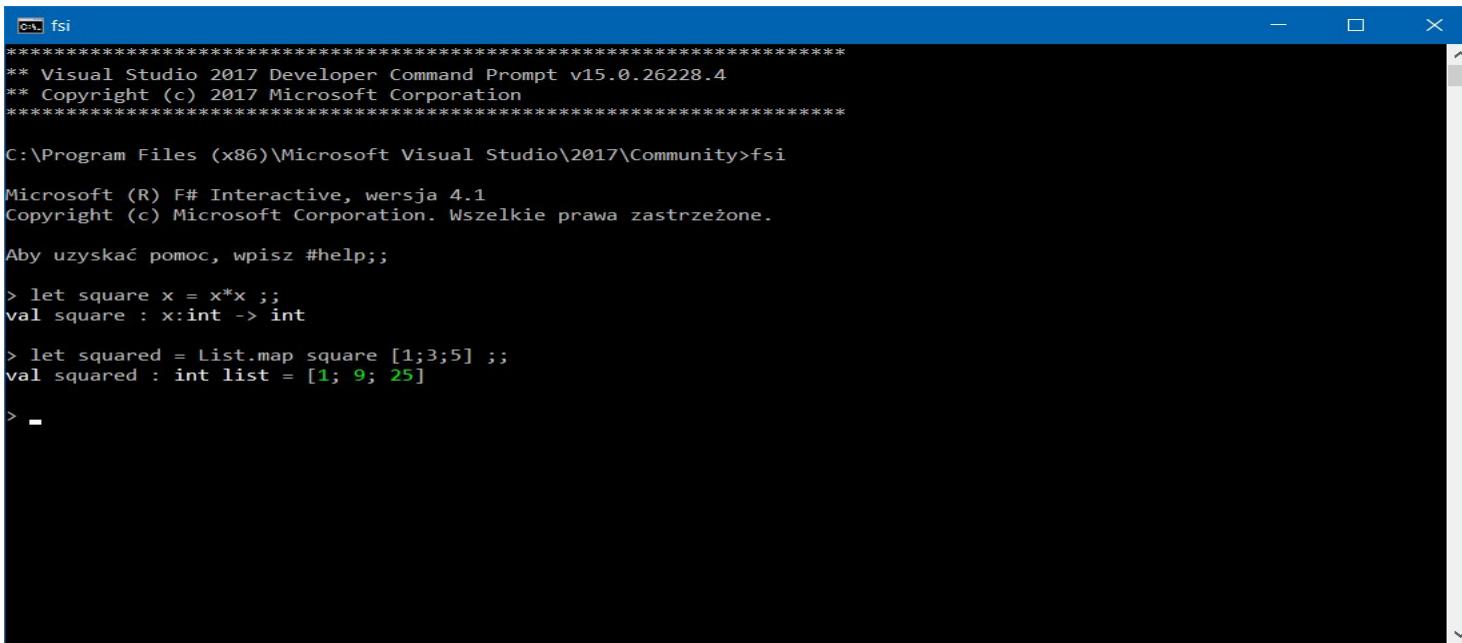
Język C#

```
class Program {
    static int Square(int x) {
        return x * x;
    }

    static void Main(string[] args) {
        List<int> input = new List<int>();
        input.Add(1);
        input.Add(2);
        input.Add(3);
        List<int> squared = new List<int>();
        for (var i = 0; i < input.Count; i++) {
            squared.Add(Square(input[i]));
        }
    }
}
```

Narzędzia F#

FSI – F# interactive (1)

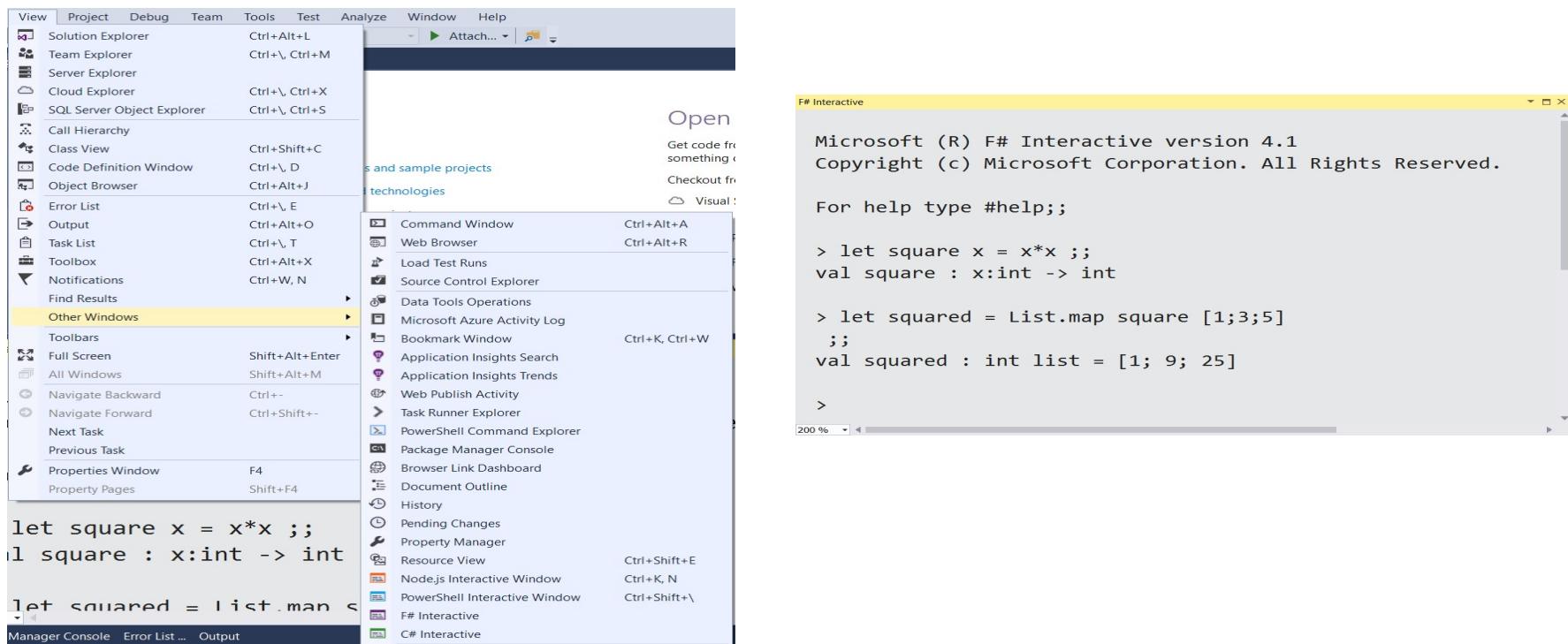


The screenshot shows the Microsoft F# Interactive window (fsi) running in a Visual Studio 2017 developer command prompt. The window title is "fsi". The console output shows the following:

```
*****  
** Visual Studio 2017 Developer Command Prompt v15.0.26228.4  
** Copyright (c) 2017 Microsoft Corporation  
*****  
  
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community>fsi  
  
Microsoft (R) F# Interactive, wersja 4.1  
Copyright (c) Microsoft Corporation. Wszelkie prawa zastrzeżone.  
  
Aby uzyskać pomoc, wpisz #help;;  
  
> let square x = x*x ;;  
val square : x:int -> int  
  
> let squared = List.map square [1;3;5] ;;  
val squared : int list = [1; 9; 25]  
> -
```

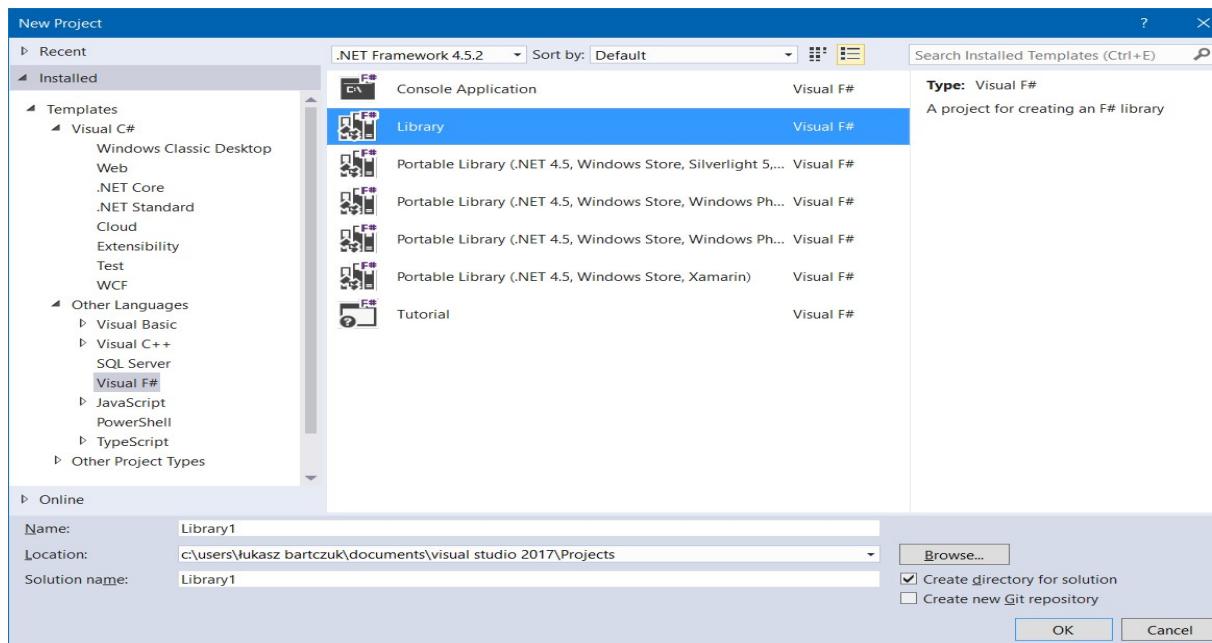
Narzędzia F#

FSI – F# interactive (2) – w Visual Studio



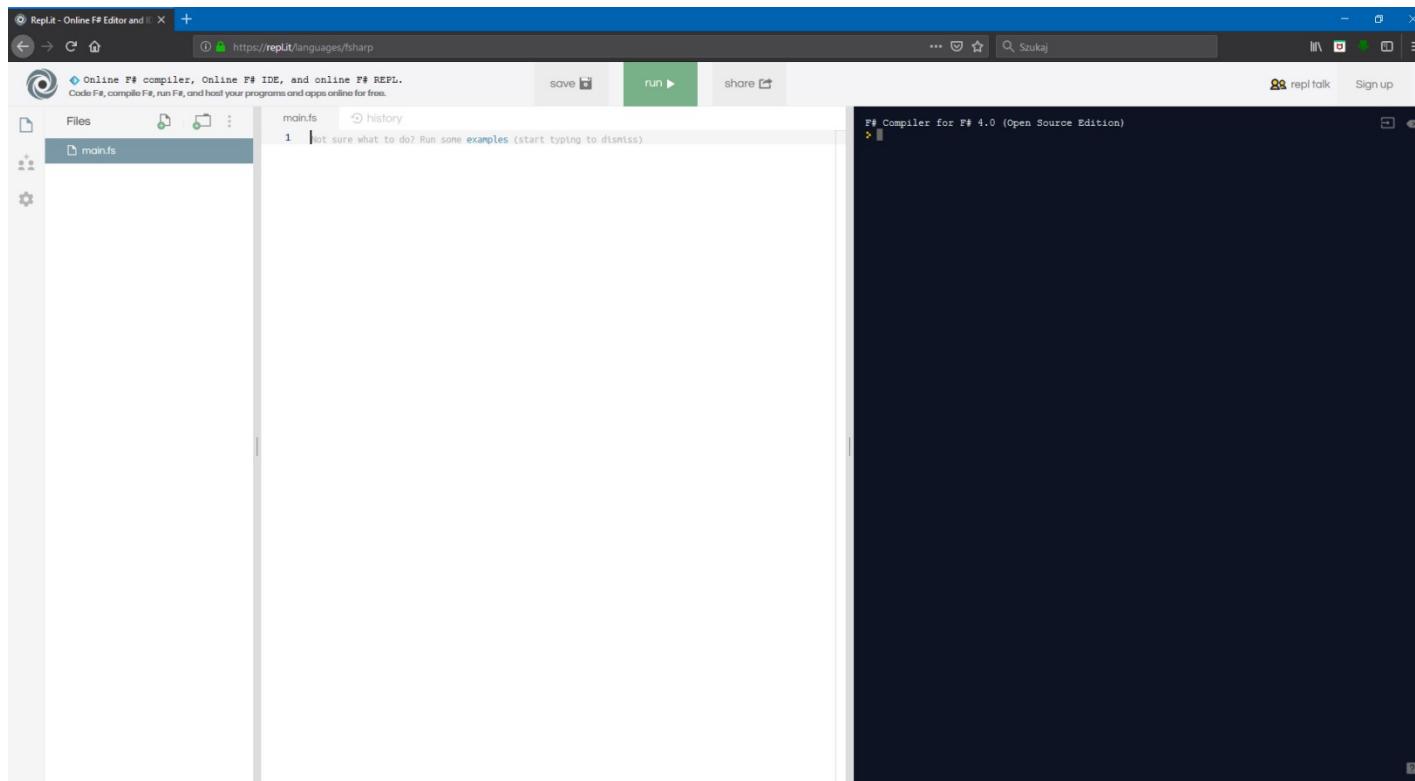
Narzędzia F#

Aplikacje F#



Narzędzia F#

<https://repl.it/languages/fsharp>



F#-deklarowanie zmiennych

Do deklarowania zmiennych służy instrukcja **let**

```
let x1 = 1.1;;
```

nazwa zmiennej

Wyrażenie, które będzie
użyte do wyznaczenia wartości

```
let x2 = 3.0+2.5*x1;;
```

F#-deklarowanie zmiennych

Zmienne stworzone za pomocą instrukcji
let są niemodyfikowalne

```
> let x = 1;;
val x : int = 1
> x = 2;;
val it : bool = false
> x <- 3;;
stdin(15,1): error FS0027: This value is not mutable.
Consider using the mutable keyword, e.g. 'let mutable x =
expression'.
```

Wartości niemodyfikowalne

Wartości niemodyfikowalne są to wartości, których składowe mogą być ustalone tylko w konstruktorze.

Wartości niemodyfikowalne



Zwiększą czytelność kodu



W pojedynczym punkcie należy sprawdzać warunki poprawności



Ułatwiają tworzenie aplikacji wielowątkowych



Wymagają większej ilości pamięci



Zwiększą zużycie procesora

F#-deklarowanie zmiennych

Funkcje w języku F# są również wartościami więc mogą być przypisane do zmiennych

```
let f = fun x -> x+1;;
```

Nazwa funkcji

Słowo kluczowe

Parametry funkcji

Ciało funkcji

Ponieważ deklaracja ta bardzo często występuje w programach funkcyjnych istnieje również jej wersja uproszczona:

```
let f x = x+1;;
```

Nazwa funkcji

Parametry funkcji

Ciało funkcji

F#-deklarowanie zmiennych

Można również deklarować funkcje wielu zmiennych:

```
let g = fun x y -> x+y;;
```

```
let g x y= x+y;;
```

Przy czym parametry funkcji oddzielamy białymi znakami i nie bierzemy w nawiasy

Jak również funkcje anonimowe:

```
fun x y -> x+y;;
```

F#-aplikowanie funkcji

Po zdefiniowaniu funkcję można aplikować do parametrów (wywoływać z parametrami)

f 1;;

funkcja

parametry

g 1 2;;

(fun x -> x+1) 1;;

F# - Zagnieżdżone deklaracje

Deklaracje let mogą zawierać kolejne deklaracje

```
let f x =  
    let y = 12  
    y+x+23;;
```

deklaracja funkcji f jednego parametru x

deklaracja zmiennej lokalnej y

wyrażenie obliczane w funkcji

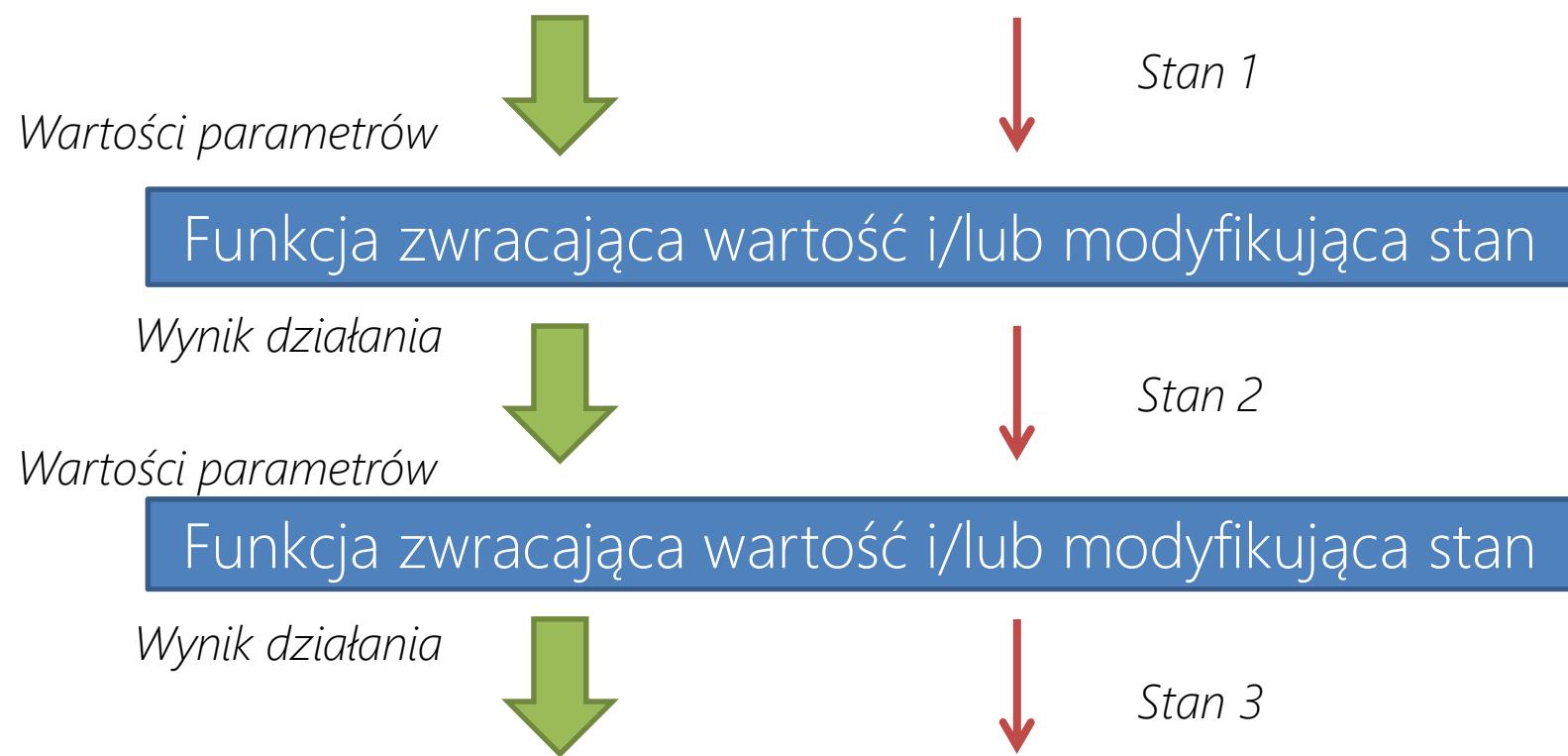
Zamiast nawiasów klamrowych liczą się wcięcia kodu

Czyste funkcje

Są to funkcje, które nie mają efektów ubocznych, a ich wynik zależy tylko od wartości parametrów



Efekty uboczne



Analiza programu



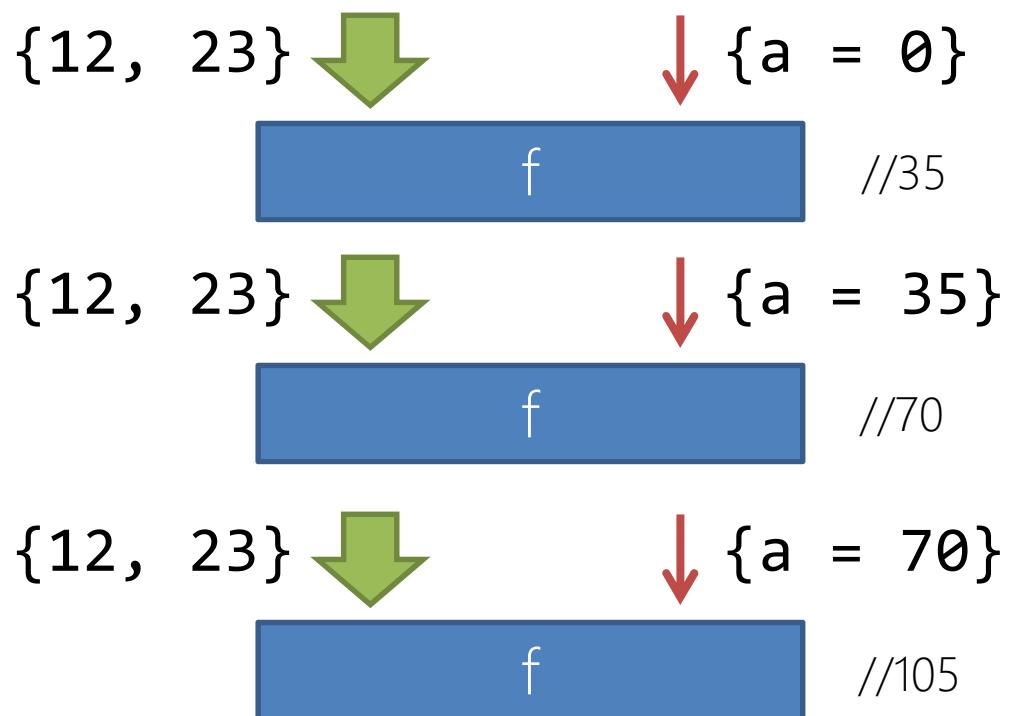
$f(12, 23)$

- Czy wynik powyższej operacji zależy tylko od parametrów?
- Czy wynik powyższej operacji będzie zawsze taki sam?

$f(12, 23) + g(34, 45)$

Analiza programu

```
int a = 0;  
  
int f(int b, int c)  
{  
    int rezultat = a+b+c;  
    a += b+c;  
    return rezultat;  
}
```



Analiza programu

```
int a = 0;
```

```
void f(int b, int c)
{
    int rezultat = a+b+c;
    a += b+c;
    return rezultat;
}
```

```
int g(int b, int c)
{
    return a+b+c;
}
```

f(12, 23) + g(34, 45)



g(34, 45) + f(12, 23)

F# - Typy danych

- F# jest językiem o ścisłym typowaniu
- F# nie wspiera konwersji niejawnych

1+2.3;;

Błąd
(próba niejawnnej konwersji
z float na int)

F# - Typy danych

Typy numeryczne

- (int, float, float32, bigint)

Łańcuchy znaków

- (string) – łączenie za pomocą operatora +

Logiczny

- (true, false)

Unit

- posiada tylko jedną wartość ()

N-tki

- podstawowy typ danych złożonych (iloczyn kartezjański)

Funkcje

- int->int

F# - typ logiczny

Wyrażenie warunkowe

```
if warunek then  
    wyrazenie_prawda  
else  
    wyrazenie_fałsz
```

```
if warunek_1 then  
    wyrazenie_prawda_1  
elif warunek_2 then  
    wyrazenie_prawda_2  
else  
    wyrazenie_fałsz
```

Podstawowe operatory logiczne: true, false, &&, ||

Podstawowe operatory relacyjne: <>, =, >, >=, <=, <

F# - N-tki

Tworzone są jako zestaw wartości (tego samego lub różnych typów danych) podanych w nawiasie i wymienionych po przecinku

```
(12, 23.4, true, "Ala ma kota");;
```

Definicja iloczynu kartezjańskiego następujących typów danych
int * float * string * bool

Aby odzyskać wartość konieczne jest napisanie specjalnych funkcji np.

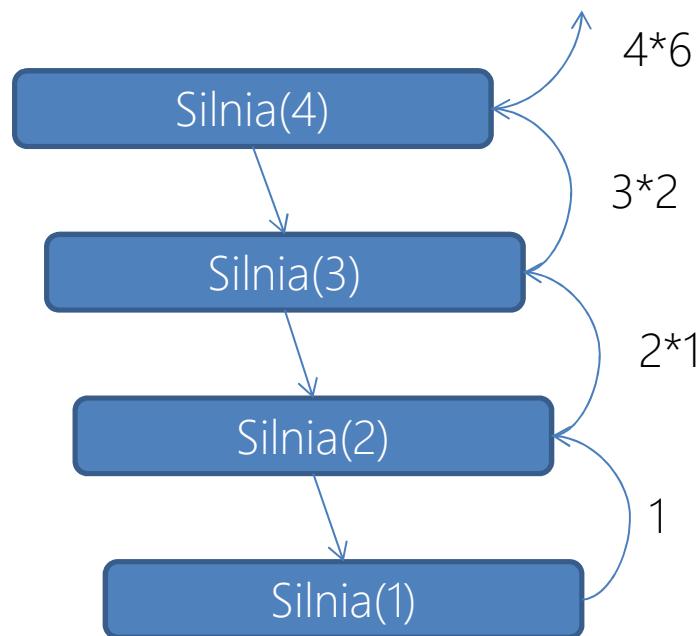
```
let second (a,b,c,d) = b;;
```

val second : a:'a * b:'b * c:'c * d:'d -> 'b

```
second (12, 23.4, true, "Ala ma kota");;
```

Rekurencja

```
private int Silnia(int n)
{
    if (n == 1)
        return 1;
    return n * Silnia(n - 1);
}
```



Rekurencja w F#

Przy deklaracjach rekurencyjnych
zamiast **let** występuje **let rec**

```
let rec silnia n =
    if n<=1 then
        1
    else
        n*silnia (n-1)
```

silnia 5
silnia 100
silnia 100000

Process is terminated due to
StackOverflowException

Rekurencja w F#

```
let rec silnia n =
  if n<=1 then
    1
  else
    n*silnia (n-1)
```

```
IL_0000: nop
IL_0001: ldarg.0      // n
IL_0002: ldc.i4.1
IL_0003: bgt.s         IL_0007
IL_0005: br.s          IL_0009
IL_0007: br.s          IL_000b
IL_0009: ldc.i4.1
IL_000a: ret
IL_000b: ldarg.0      // n
IL_000c: ldarg.0      // n
IL_000d: call           int32 L1.P::silnia(int32)
IL_0012: mul
IL_0013: ldc.i4.1
IL_0014: sub
IL_0015: ret
```

Rekurencja w F#

```
let rec silnia n =
    if n<=1 then
        1
    else
        n*silnia (n-1)
```

```
public static class Przyklad
{
    public static int silnia(int n)
    {
        if (n <= 1)
            return 1;
        return n * Przyklad.silnia(n - 1);
    }
}
```

Rekurencja

$$fib(n) = \begin{cases} 1 & \text{jeżeli } n = 1 \text{ lub } n = 2 \\ fib(n - 1) + fib(n - 2) & \text{jeżeli } n > 1 \end{cases}$$



Ille czasu potrwa wyznaczenie Fib(80)
że jedno wywołanie funk

Fib(80) = 23416728
Fib(200) = 17340252117279781315



Rekurencja – Fib(200)

$T \approx 5498557875849753080913401.74$ [lat]

Wiek wrzechświata $\approx 13,8$ miliarda lat

Rekurencja z akumulatorem

```
let rec silniaAcc n acc =  
  if n<=1 then  
    acc  
  else  
    silniaAcc (n-1) (acc*n)
```

Wywołanie rekurencyjne jest ostatnią instrukcją w tej funkcji

```
silniaAcc 5 1;;
```

IL_0000:	nop	
IL_0001:	ldarg.0	// n
IL_0002:	ldc.i4.1	
IL_0003:	bgt.s	IL_0007
IL_0005:	br.s	IL_0009
IL_0007:	br.s	IL_000b
IL_0009:	ldarg.1	// acc
IL_000a:	ret	
IL_000b:	ldarg.0	// n
IL_000c:	ldc.i4.1	
IL_000d:	sub	
IL_000e:	ldarg.1	// acc
IL_000f:	ldarg.0	// n
IL_0010:	mul	
IL_0011:	starg.s	acc
IL_0013:	starg.s	n
IL_0015:	br.s	IL_0000

Rekurencja z akumulatorem

```
let rec silniaAcc n acc =  
  if n<=1 then  
    acc  
  else  
    silniaAcc (n-1) (acc*n)
```

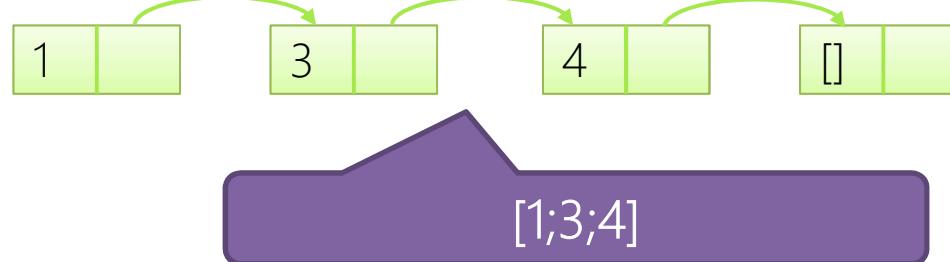
Wywołanie rekurencyjne jest ostatnią instrukcją w tej funkcji

```
public static  
int silniaAcc(int n, int acc)  
{  
  int num;  
  for (; n > 1; n = num) {  
    num = n - 1;  
    acc *= n;  
  }  
  return acc;  
}
```

Rekurencja z akumulatorem

```
let silnia n =
    let rec silniaAcc n acc =
        if n<=1 then
            acc
        else
            silniaAcc (n-1) (acc*n)
    silniaAcc n 1
```

F# - Listy (1)



- Typ danych **T list**, gdzie **T** jest dowolnym typem danych języka F#
- Listy są tworzone za pomocą operatora `[]` oraz `::`
- `[]` oznacza pustą listę
- `[1;3;4]` – oznacza listę liczb całkowitych 1,3,4,[]

F# - Listy (2)

- Listy można również tworzyć za pomocą operatora ::
- **x::xs** oznacza listę z pierwszym elementem **x** typu **T** a, a **xs** typu **T list**

Głowa listy

x::xs

Ogon listy

[1;3;4]

1::3::4::[]

Te zapisy stworzą listy ekwiwalentne

1::3::4

Zapis błędny

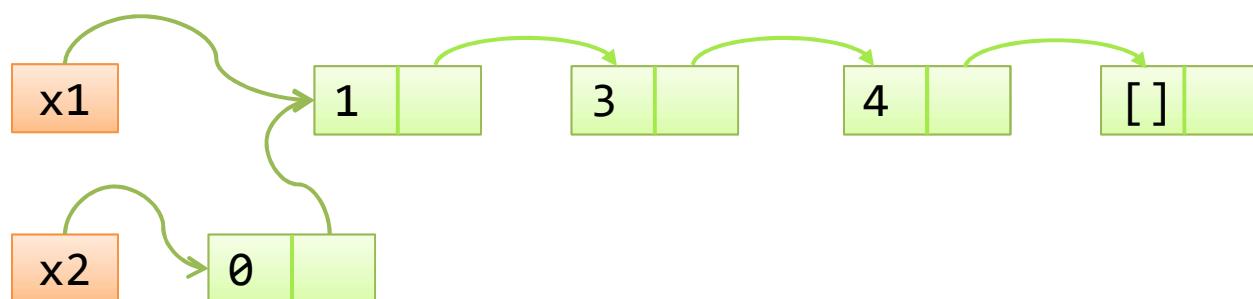
- Listy są niemodyfikowalne. Operator :: tworzy nową listę zamiast modyfikować istniejącą

F# - Listy (3)

`let x1 = 1::3::4::[]`



`let x2 = 0::x1`



`LanguagePrimitives.PhysicalEquality
(List.tail x2) x1;;`

F# - Listy (4)

Listy mogą być tworzone również za pomocą zakresów:

```
let x = [1..10];;
```

[1;2;3;4;5;6;7;8;9;10]

```
let x = [1..2..10];;
```

[1;3;5;7;9]

```
let x = ['a'..'e'];;
```

['a','b','c','d','e']

Skracanie list (1)

- Jest to mechanizm pozwalający na definiowanie list z użyciem istniejących list

Iterator, który przechodzi po elementach innej listy

Lista wyjściowa

```
let parzyste = [for x in 1..10 do  
                if x%2 = 0 then yield x]
```

Warunek pozwalający filtrować listę

Operator dodawania wybranego elementu do nowej listy

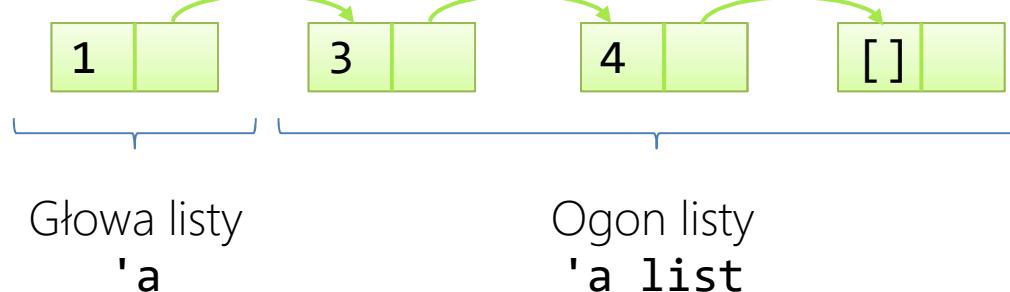
Skracanie list (2)

W ramach jednej operacji tworzenia list można wykorzystać wiele operatorów np..

```
let dzielniki n = [for x in 1 .. n do  
                   if n%x = 0 then yield x]
```

```
let wspolneDzielniki m n = [for x in dzielniki m do  
                            for y in dzielniki n do  
                            if x=y then yield x]
```

Operacje działające na listach



$$lista = \begin{cases} [] & \text{gdy lista jest pusta} \\ x :: xs & \text{gdy lista ma głowę i ogon} \end{cases}$$

Operacje działające na listach

```
let rec liczbaElementow l =
    if List.isEmpty l then
        0
    else
        1 + liczbaElementow (List.Tail l)
```

F# - Dopasowywanie wzorców

Wyrażenie **match** pozwala dopasować wzorzec do wartości

Funkcja o nazwie i przyjmująca parę dwóch elementów bool

```
let i para =  
    match para with  
        | (false, _) -> false  
        | (true, x) -> x
```

Pierwszy wzorzec (pierwszy element o wartości false, drugi obojętnie jaki)

Drugi wzorzec (pierwszy element o wartości true, a wartość drugiego ma być podstawiona pod zmienną x)

```
let i = function  
    | (false, _) -> false  
    | (true, x) -> x
```

Liczy się pierwsze dopasowanie

Lista wyrażeń musi być kompletna dla danego typu danych

F# - Dopasowywanie wzorców

```
let i = function
| (false, _) -> false
| (true, x) -> x
```

```
let pierw (a,b) = a;;
let drugi (a,b) = b;;

let i para =
  if (pierw para) = false then
    false
  else
    (drugi para);;
```

Oba programy robią to samo

Przykładowe funkcje operujące na listach (1)

```
let rec LiczbaElementow = function
| [] -> 0
| x::xs -> 1+LiczbaElementow xs
```

```
let rec DodajNaKoniec lista wartosc =
  match lista with
  | [] -> wartosc::[]
  | x::xs -> x::(DodajNaKoniec xs wartosc)
```

Przykładowe funkcje operujące na listach (2)

```
let rec PolaczListy xs ys =
    match xs with
    | [] -> ys
    | x::xss -> x::(PolaczListy xss ys)
```

```
PolaczListy [1;3;4] [2;3;4]
```

Łączenie list jest operacją tak popularną, że w F# istnieje do tego specjalny operator @

```
[1;3;4] @ [2;3;4]
```

Przykładowe funkcje operujące na listach (3)

Uwaga na operator @! Jest wolny!

```
let l1 = [1;2;3];;
```



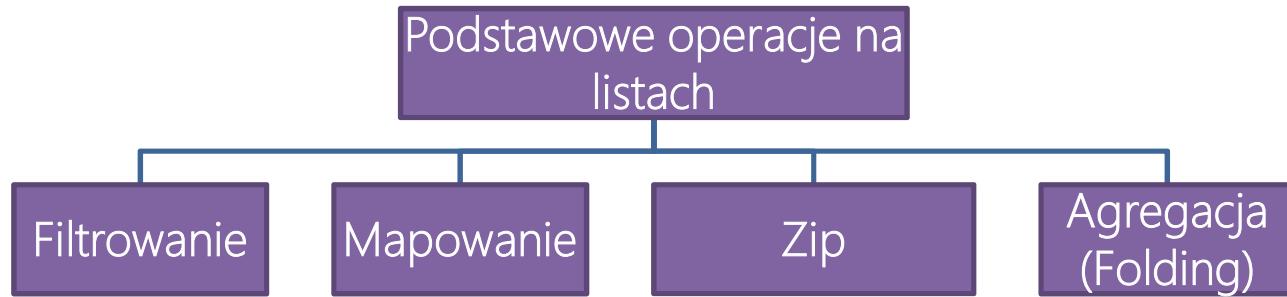
```
let l2 = [4;5;6];;
```



```
let l3 = l1@l2;;
```



Podstawowe operacje na listach



Funkcje wyższych rzędów

```
let f [x y] =>
```

To mogą być dane lub funkcje

...

```
a
```

To mogą być dane lub funkcje

Przykład

```
let rec iloczyn lista =
  if List.isEmpty lista then
    1.0
  else
    lista.Head * iloczyn lista.Tail
```

```
let rec suma lista =
  if List.isEmpty lista then
    0.0
  else
    lista.Head + suma lista.Tail
```

```
let listaCalk = [1..10];;
```

```
let il = iloczyn
      listaCalk;;
```

```
let su = suma
      listaCalk;;
```

Przykład

```
let rec iloczyn lista =  
    if List.isEmpty lista then  
        1.0  
    else  
        lista.Head * iloczyn lista.Tail
```

Wartość początkowa

```
let rec suma lista =  
    if List.isEmpty lista then  
        0.0  
    else  
        lista.Head + suma lista.Tail
```

Działanie

Jakie są elementy
wspólne tych
dwóch funkcji?



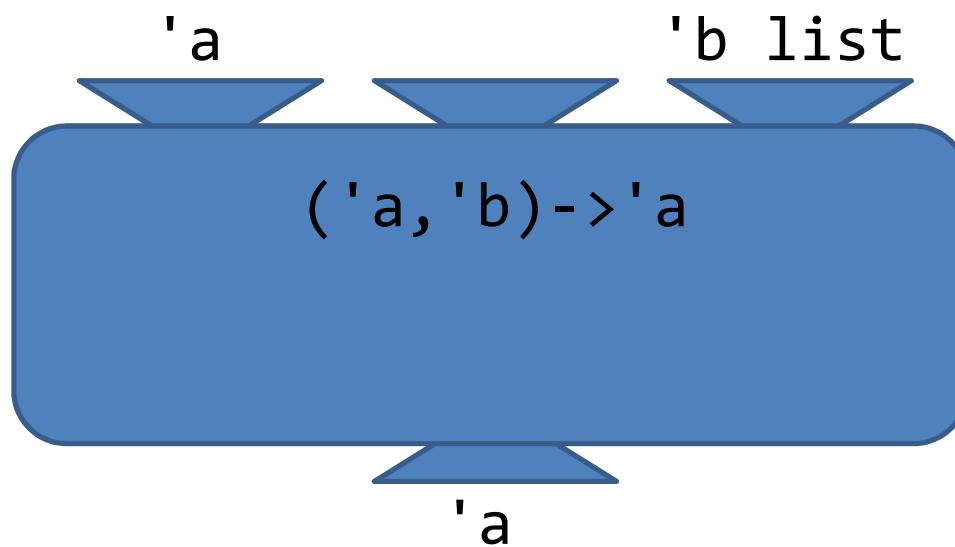
Przykład

$'a \quad ('a, 'b) \rightarrow 'a$

```
let rec agregacja pocz funkcja lista =
    if List.isEmpty lista then
        pocz
    else
        funkcja (agregacja pocz funkcja lista.Tail) lista.Head
```

Przykład

```
let rec agregacja pocz funkcja lista =  
    if List.isEmpty lista then  
        pocz  
    else  
        funkcja (agregacja pocz funkcjalista.Tail) lista.Head
```



Przykład

```
let rec agregacja pocz funkcja lista =
    if List.isEmpty lista then
        pocz
    else
        funkcja (agregacja pocz funkcjalista.Tail) lista.Head
```

```
let f1 a x = a+x;;
let f2 a x = a*x;;

agregacja 0 f1 lista;;
agregacja 1 f2 lista;;
```

Przykład

```
let rec agregacja pocz funkcja lista =
    if List.isEmpty lista then
        pocz
    else
        funkcja (agregacja pocz funkcjalista.Tail) lista.Head
```

```
let f1 a x = a+x;;
let f2 a x = a*x;;
agregacja 0 f1 lista;;
agregacja 1 f2 lista;;
```

```
agregacja 0 (+) lista;;
agregacja 1 (*) lista;;
```

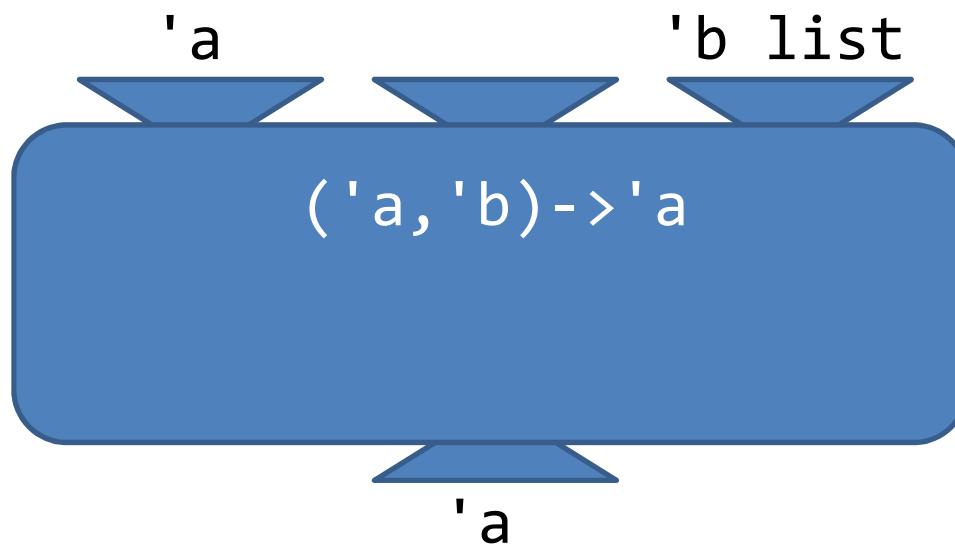
Przykład

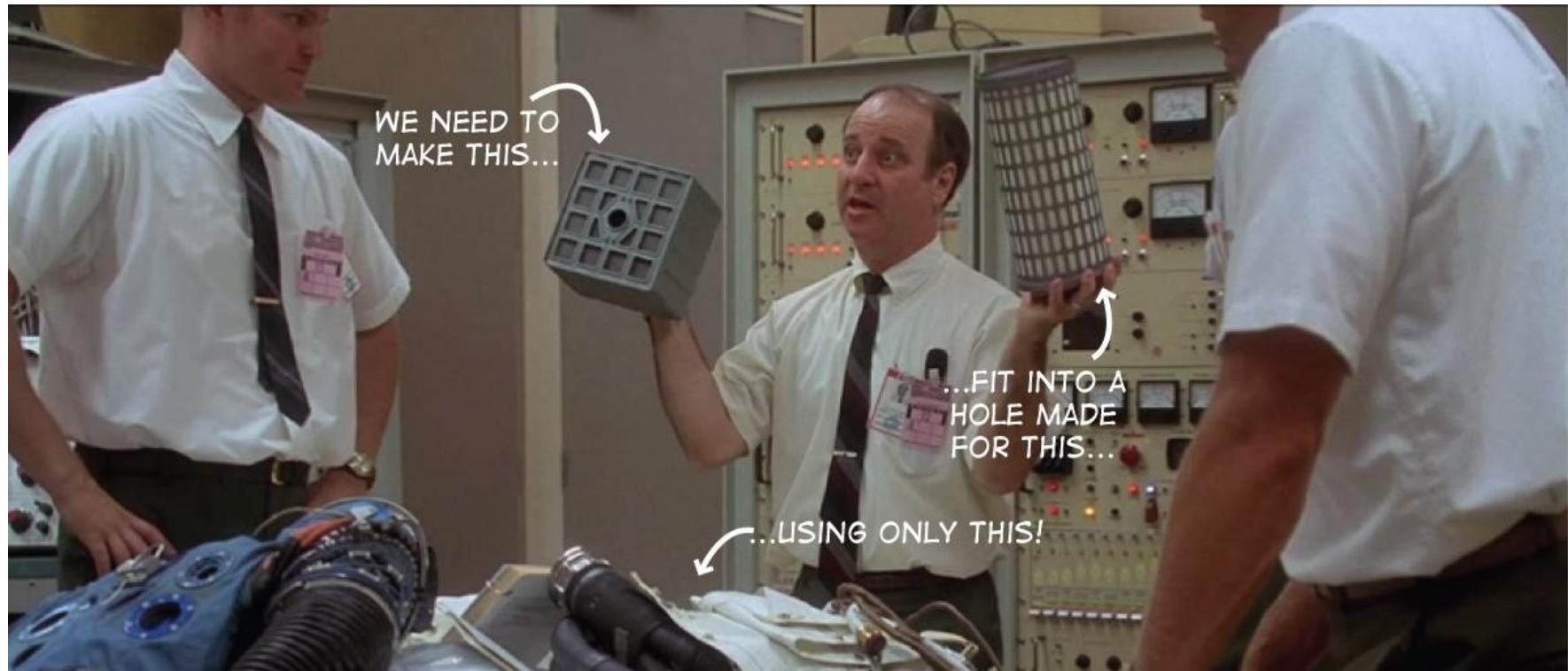
```
let rec agregacja pocz funkcja lista =
    if List.isEmpty lista then
        pocz
    else
        funkcja (agregacja pocz funkcjalista.Tail) lista.Head
```

```
agregacja 0 (fun a x -> if x%2 = 0 then a+x else a) lista
agregacja 0 (fun a x -> if x%3 = 0 then a+x else a) lista
```

Przykład

```
let f1 a x = if x%2 = 0 then a+x else a  
let f2 a x = if x%3 = 0 then a+x else a  
  
let f3 a x v = if x%v = 0 then a+x else a
```





Przykład

$'a \ 'b \ -> \ 'a$



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

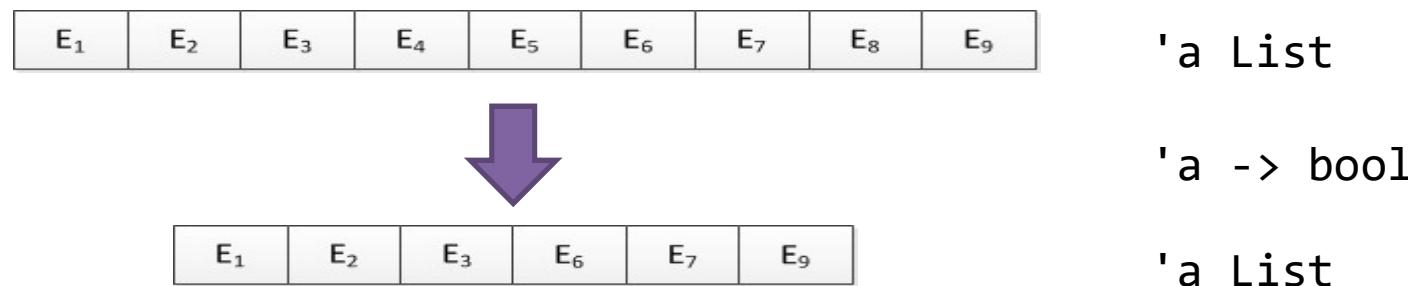
```
agregacja 0 (pred 2) lista;;
agregacja 0 (pred 3) lista;;
```

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

```
let pred v = fun a x -> f3 a x v
```

Filtrowanie

Tworzenie nowej listy na podstawie istniejącej, przy czym wybierane są tylko te elementy, które spełniają określony warunek.

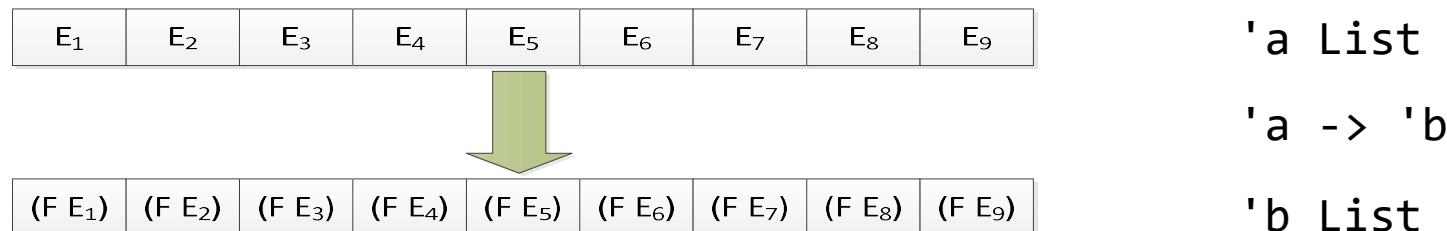


Operacja ta może być zrealizowana za pomocą skracania list

```
let filtr f lista =  
  [for e in lista do if f(e) then yield e];;
```

Mapowanie

Polega na utworzeniu nowej listy stosując do każdego elementu pewną funkcję

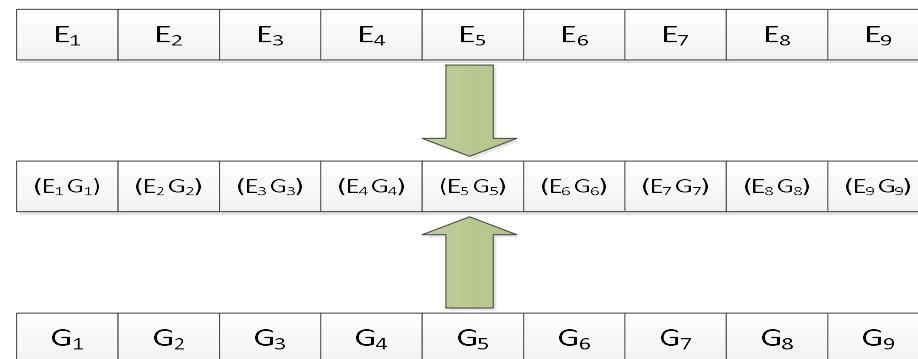


```
let l1 = [for i in [1;2;3;4;5] do yield x+1];;
```

```
let map f lista =
    [ for e in lista do yield f(e)];;
```

zip

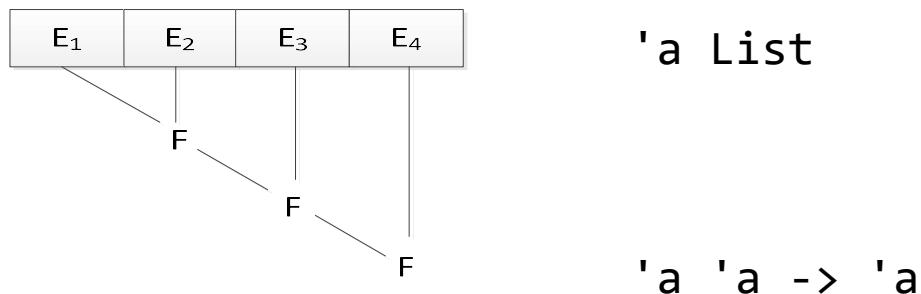
Polega na pobraniu dwóch list i stworzeniu trzeciej składającej się z par



```
let rec zip xs ys =
  match (xs, ys) with
  | ([], []) -> []
  | (x::xs, []) -> failwith "Niezgodne dlugosci"
  | ([], y::ys) -> failwith "Niezgodne dlugosci"
  | (x::xs, y::ys) -> (x,y)::(zip xs yss);;
```

Agregacja

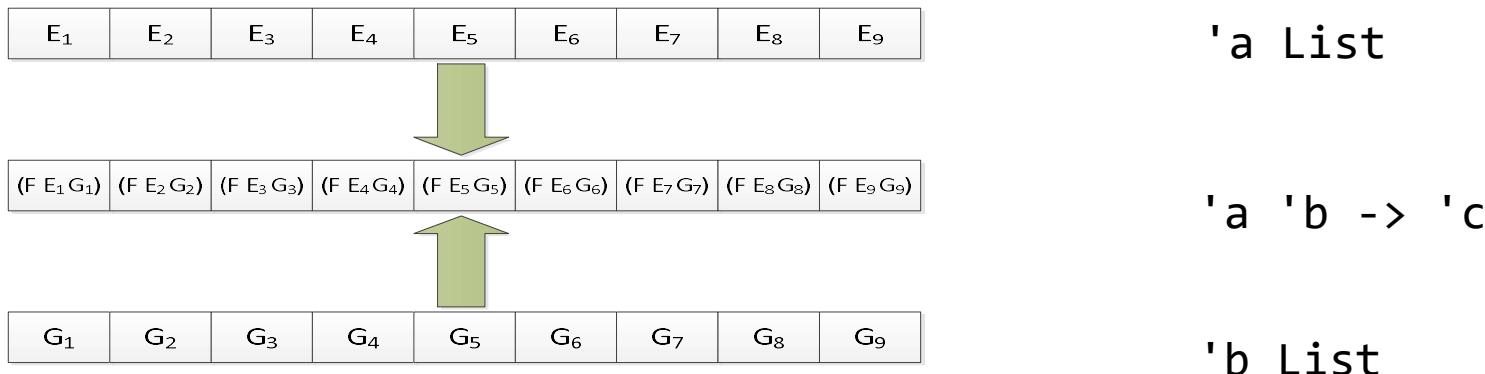
Jest to operacja zamieniająca listę elementów na wartość skalarną



```
let rec agregacja f xs =
  match xs with
  | [] -> failwith "za mało elementów"
  | (x1::[]) -> failwith "za mało elementów"
  | (x1::x2::[]) -> f x1 x2
  | (x::xss) -> (f x (agregacja f xss))
```

Połączenie

Połączenie jest operacją podobną do mapowania, ale działającą na wielu listach



```
let rec polacz f xs ys =
  match (xs, ys) with
  | ([], []) -> []
  | (x::xss, []) -> failwith "Niezgodne dlugosci"
  | ([], y::yss) -> failwith "Niezgodne dlugosci"
  | (x::xss, y::yss) -> (f x y)::(polacz f xss yss);;
```

Inne operacje na listach

```
let dlugosc xs = agregacja (fun x y->x+y)
                           (map (fun x-> 1) xs);;
```

```
let ile cond xs = agregacja (fun x y->x+y)
                       (map (fun x->1) filtr(cond xs));;
```

```
let min xs = agregacja (fun x y -> if x<y then x else y) xs;;
```

```
let max xs = agregacja (fun x y -> if x>y then x else y) xs;;
```

Currying

Umożliwia traktowanie funkcji wielu parametrów
jako funkcji jednego parametru

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

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

Currying

Umożliwia traktowanie funkcji wielu parametrów
jako funkcji jednego parametru

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

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

→ Powstaje nowa funkcja
jednego parametru

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

Currying

```
let rec agregacja pocz funkcja lista =
    if List.isEmpty lista then
        pocz
    else
        funkcja (agregacja pocz funkcjalista.Tail) lista.Head
```

```
let agregujPodzielne a x v = if x%v = 0 then a+x else a
let pred v = fun a x -> agregujPodzielne a x v
```

Currying

```
let rec agregacja pocz funkcja lista =
    if List.isEmpty lista then
        pocz
    else
        funkcja (agregacja pocz funkcjalista.Tail) lista.Head
```

zmieńmy kolejność parametrów

```
let agregujPodzielne v a x = if x%v = 0 then a+x else a
let pred = agregujPodzielne 2
```

Currying

```
let rec agregacja pocz funkcja lista =
    if List.isEmpty lista then
        pocz
    else
        funkcja (agregacja pocz funkcjalista.Tail) lista.Head
```

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

agregacja 0 (agregujPodzielne 2) lista;;
agregacja 0 (agregujPodzielne 3) lista;;
```

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
```

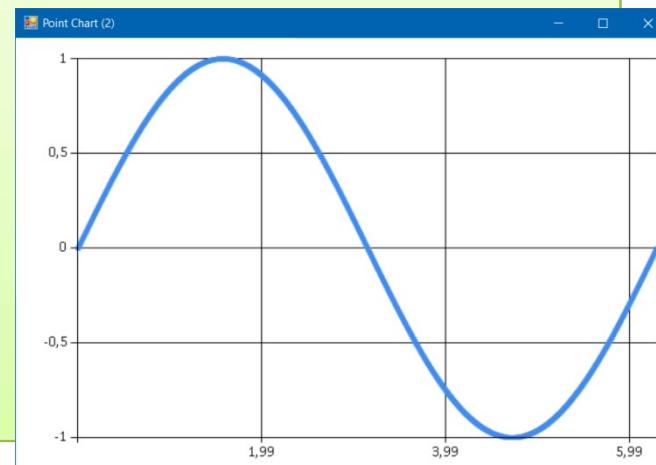
Operator pipe |>

```
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 lista =
    if List.isEmpty lista then
        pocz
    else
        funkcja (agregacja pocz funkcjalista.Tail) lista.Head
```

```
lista |> agregacja 0 (agregujPodzielne 3) ;;
```

Podwójny i potrójny pipe

(1, 2) | > min

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

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;;
```

F# - rekordy

```
type Osoba = {  
    Imie : string;  
    Nazwisko : string  
}
```

Składowe rozdzielamy nową linią
lub średnikiem

```
val osoba1 : Osoba = {Imie = "Tomek";  
                      Nazwisko = "Kowalski";}
```

```
let osoba1 = {Imie = "Tomek"; Nazwisko = "Kowalski"}  
let osoba2 = {Imie = "Tomek"; Nazwisko = "Kowalski"}  
let osoba3 = {Imie = "Ewa"; Nazwisko = "Nowak"}
```

```
osoba1 = osoba2 //true  
osoba1 = osoba3 //false
```

F# - rekordy

Rekordy są niemodyfikowalne

```
let osoba3 = {Imie = "Ewa"; Nazwisko = "Nowak"}
```

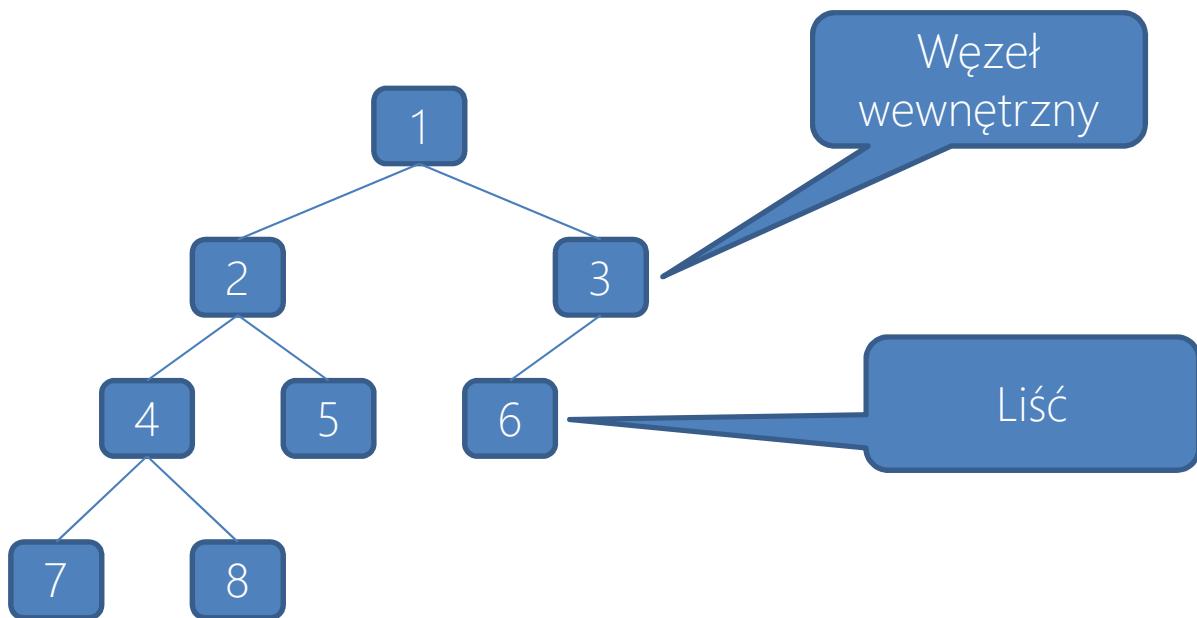
```
let osoba4 = {osoba2 with Nazwisko="Adamski"}
```

F# - rekordy

Do rekordów można dołączać metody

```
type Osoba = {  
    Imie : string;  
    Nazwisko : string  
}  
  
type Osoba with  
    member this.ZNazwiskiem(nazwisko) =  
        { this with Nazwisko = nazwisko }
```

Struktury drzewiaste



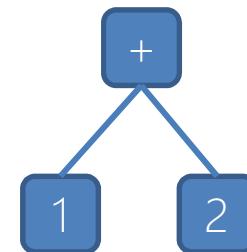
$$Drzewo = \left\{ \begin{matrix} n \\ (n * Drzewo * Drzewo) \end{matrix} \right.$$

Unie z dyskryminatorem

```
type Operator =  
| Plus  
| Minus
```

Różne wersje tego typu

```
type Drzewo =  
| Operator of Operator*Drzewo*Drzewo  
| Wartosc of float
```



```
let jeden_plus_dwa = Operator(Plus, Wartosc(1.0), Wartosc(2.0))
```

Unie z dyskryminatorem

```
let rec LiczWezly = function
| Wartosc x -> 1
| Operator (op, lewo, prawo) ->
    1
    + (LiczWezly lewo)
    + (LiczWezly prawo)
```

Obliczanie wartości wyrażeń

```
let rec oblicz = function
| Operator (op, lewo, prawo) ->
  match op with
  | Plus -> oblicz lewo + oblicz prawo
  | Minus -> oblicz lewo - oblicz prawo
| Wartosc w -> w
```

Obliczanie wartości wyrażeń

```
type Drzewo with
    member this.oblicz =
        match this with
        | Operator (op, lewo, prawo) ->
            match op with
            | Plus -> lewo.oblicz + prawo.oblicz
            | Minus -> lewo.oblicz - prawo.oblicz
        | Wartosc wartosc -> wartosc
```

Kontynuacje

Kontynuacje są to funkcje, które jako parametr przyjmują wyniki poprzednich obliczeń

Funkcja standardowa

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

Funkcja z kontynuacją

```
let dodajZKontynuacją a b f = f (a+b)
```

kontynuacja

Rekurencja z kontynuacją

```
let rec silnia n kont =
  if n<=1 then
    kont 1
  else
    silnia (n-1) (fun x-> kont(n*x))
```

```
silnia 5 (fun x->x)
silnia 5 (fun x->x+12)
silnia 5 (fun x->printfn "%d" x)
```

Rekurencja z kontynuacją

```
let rec silnia n kont =
  if n<=1 then
    kont 1
  else
    silnia (n-1) (fun x-> kont(n*x))
```

```
silnia 3 (fun x->x)
```

```
silnia 2 (fun x-> (fun x->x)(3*x))
```

```
silnia 1 (fun x-> (fun x-> (fun x->x)(3*x))(2*x))
```

```
(fun x-> (fun x-> (fun x->x)(3*x))(2*x)) 1
```

```
(fun x-> (fun x->x)(3*x))(2*1)
```

```
(fun x->x)(3*2)
```

Rekurencja z kontynuacją

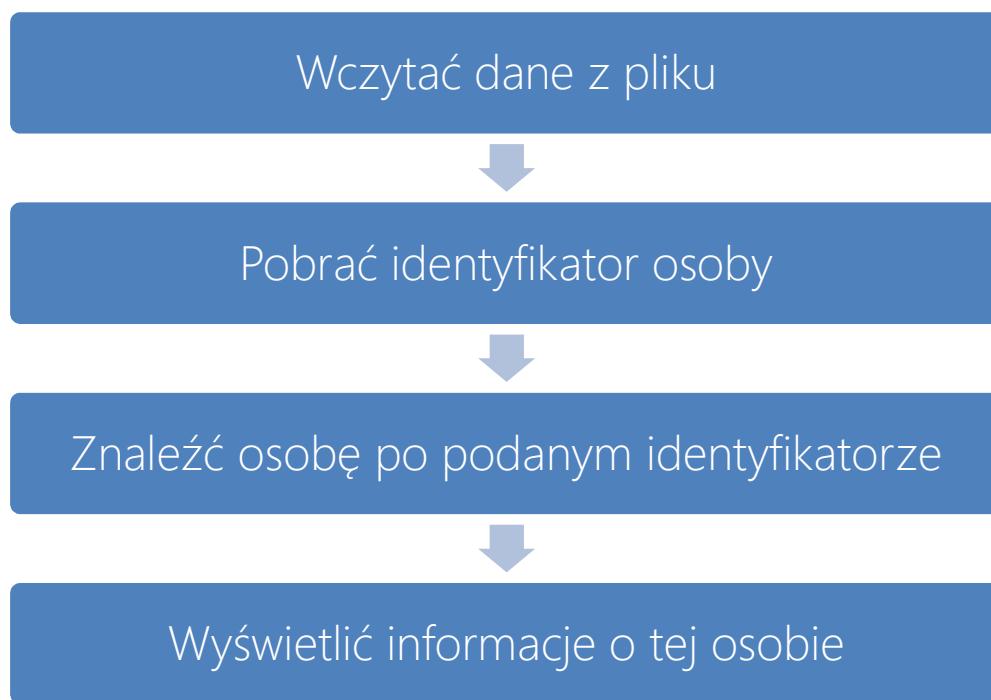
```
let rec liczbaElementow ls kont =
  match ls with
  | [] -> kont 0
  | _::xs -> liczbaElementow xs (fun suma -> (kont suma+1))
```

```
liczbaElementow [1;2;3] (fun x->x);;
```

Obliczanie wartości wyrażeń

```
let rec LiczWezly kont = function
| Wartosc x -> kont 1
| Operator (op, lewo, prawo) ->
  LiczWezly (fun lL-> LiczWezly (fun lP -> lL+lP+1) prawo ) lewo
```

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) =
    Console.WriteLine osoba.id
    Console.WriteLine osoba.imie
    Console.WriteLine 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"
    Console.WriteLine "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"
    Console.WriteLine "Podaj id osoby: "
    Console.ReadLine ()
    |> int
    |> znajdzOsobePoId dane
    |> wyswietlDane
```

Co w tej aplikacji może pójść źle?

```
let main argv =
    let dane = wczytajDane "dane.json"
    Console.WriteLine "Podaj id osoby: "
    Console.ReadLine ()
|> int
|> znajdzOsobę
|> wyswietlOsobę
```



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"
    Console.WriteLine "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"  
    Console.WriteLine "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"  
    Console.WriteLine "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"  
    Console.WriteLine "Podaj id osoby:"  
    Console.ReadLine ()  
    |> int  
    |> znajdzOsobePoId dane  
    |> wyswietlDane
```

Jeżeli osoba będzie null to będzie wyjątek

```
let wyswietlDane (osoba:Osoba) =  
    Console.WriteLine osoba.id  
    Console.WriteLine osoba.imie  
    Console.WriteLine osoba.nazwisko
```

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
```

Naprawa aplikacji

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

Naprawa aplikacji

```
let pokazBlad (wiadomosc:string) =
    Console.ForegroundColor <- ConsoleColor.Red
    Console.WriteLine wiadomosc
    Console.ResetColor();
```

```
let rec wczytajId () =
    try
        Console.WriteLine "Podaj id osoby: "
        Console.ReadLine () |> int
    with
    | _ -> Console.Clear();
              pokazBlad "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
            pokazBlad "Nie ma osoby o takim id"
    with
    | _ -> pokazBlad "Tego błędu już nie naprawię"
0
```

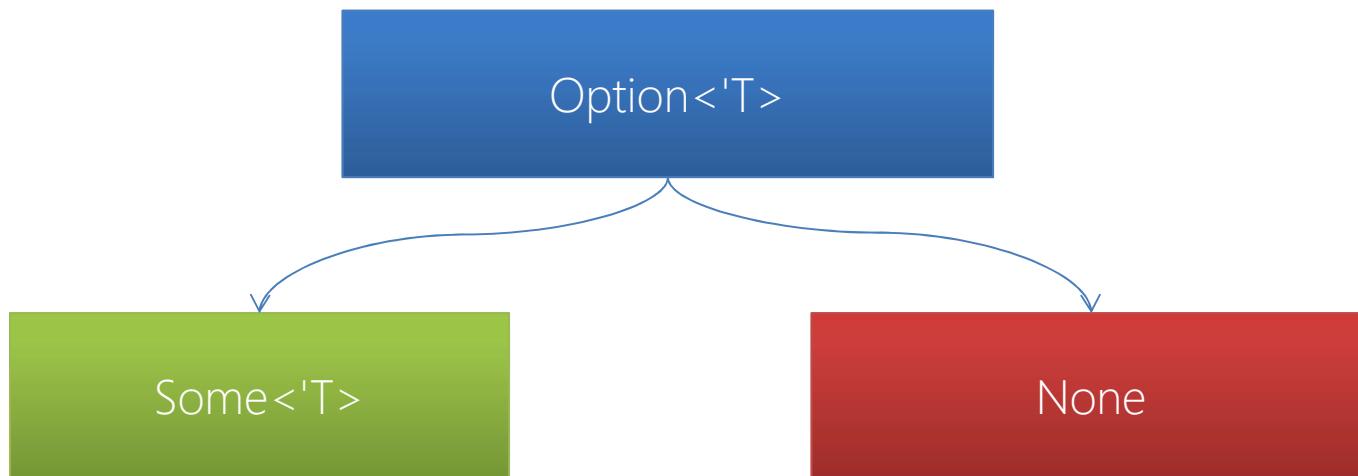


Czy można to napisać lepiej?

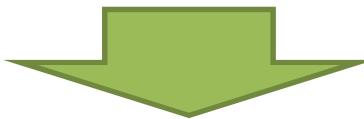
OBSŁUGA WARTOŚCI NULL

Opcje

Typ Option pozwala na określanie czy dana wartość istnieje czy nie



```
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
```



```
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
```

```
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
        Console.WriteLine "Podaj id osoby: "
        Console.ReadLine () |> int
    with
        | _ -> Console.Clear();
                  pokazBlad "Błędny identyfikator użytkownika"
                  wczytajId ()
```

Opcje w tworzeniu pętli

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

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

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

```
[<EntryPoint>]
let main argv =
    try
        let dane = wczytajDane "dane.json"
        loop wczytajId
            |> znajdzOsobePoId2 dane
            |> pokazWynik
        with
        | _ -> pokazBlad "Tego błędu już nie naprawię"
        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 =
    try
        let dane = wczytajDane "dane.json"
        loop wczytajId
        |> znajdzOsobePoId dane
        |> pokazWynik
    with
    | _ -> pokazBlad "Tego błędu już nie naprawię"
0
```

string → Osoba list option

Nie
pasuje

Osoba list → Osoba option

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

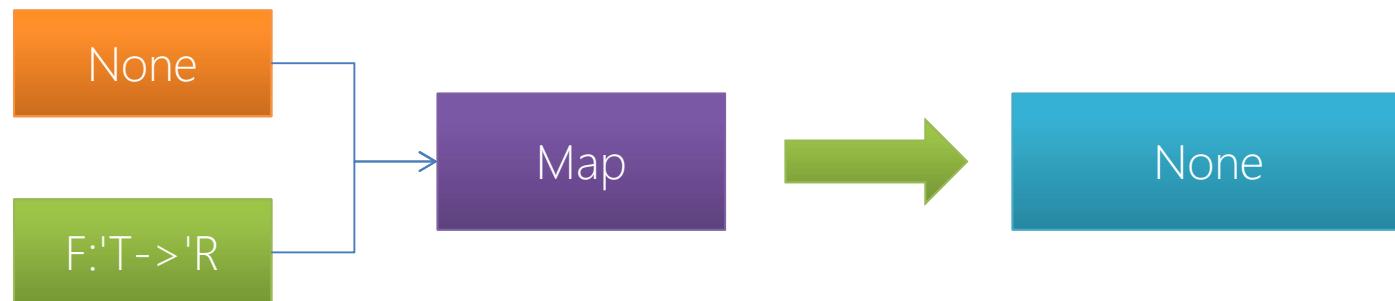
MAPOWANIE, FUNKTORY I MONADY

Mapowanie

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

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

Mapowanie



Mapowanie

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

Funktory

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

W programowaniu funkcyjnym dowolny typ wyposażony w funkcję Map nazywamy funktem

Mapowania

```
let naString (osoba:Osoba) =
    osoba.imie+ " "+osoba.nazwisko

[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane.json"
    Option.iter (fun dane ->
        loop wczytajId
        |> znajdzOsobePoId dane
        |> map naString
        |> Console.WriteLine
    ) dane
    0
```

Osoba list
→
unit

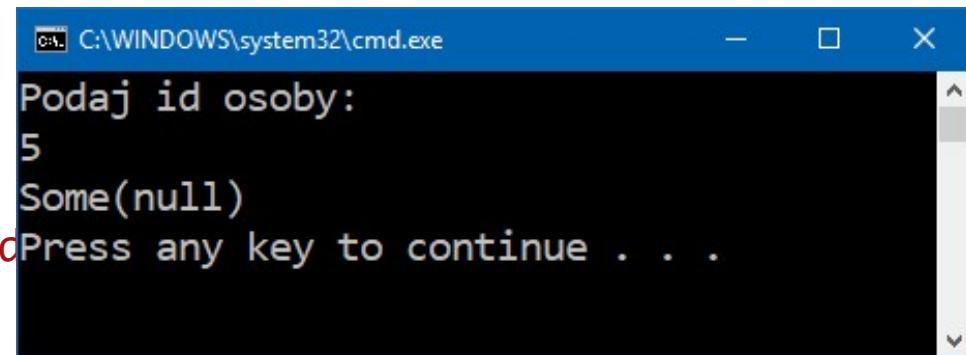
Mapowania

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

Osoba list
→
string option

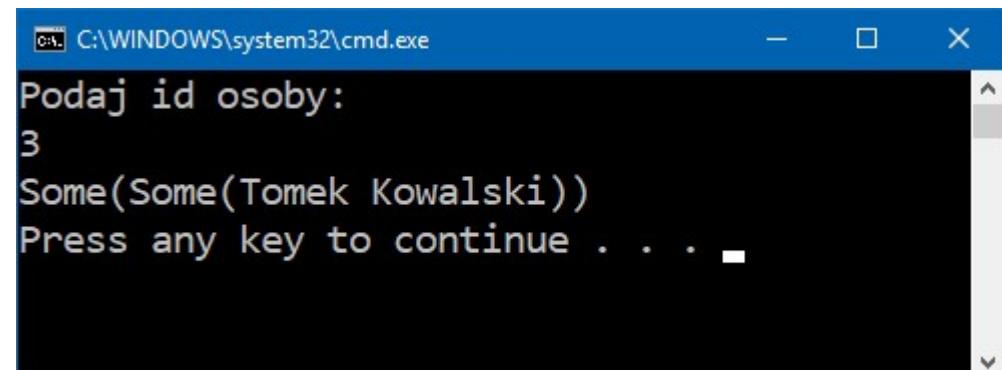
Mapowania

```
[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane.txt"
    map (fun dane ->
        loop wczytajId
        |> znajdzOsobePoId dane
        |> map naString
    ) dane
    |> Console.WriteLine
0
```



C:\WINDOWS\system32\cmd.exe

Podaj id osoby:
5
Some(null)
Press any key to continue . . .

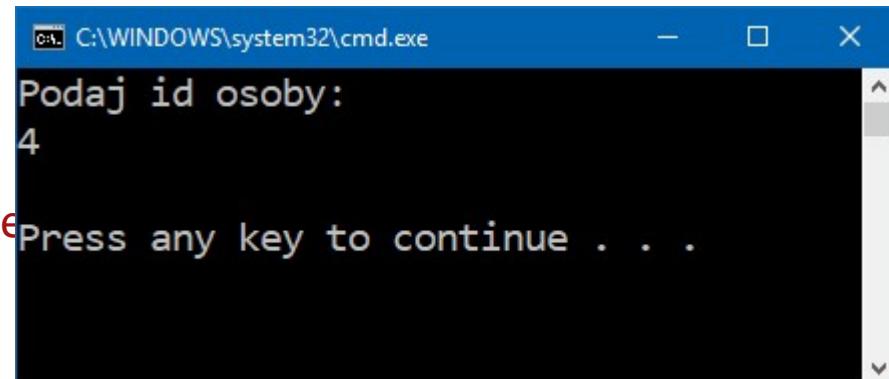


C:\WINDOWS\system32\cmd.exe

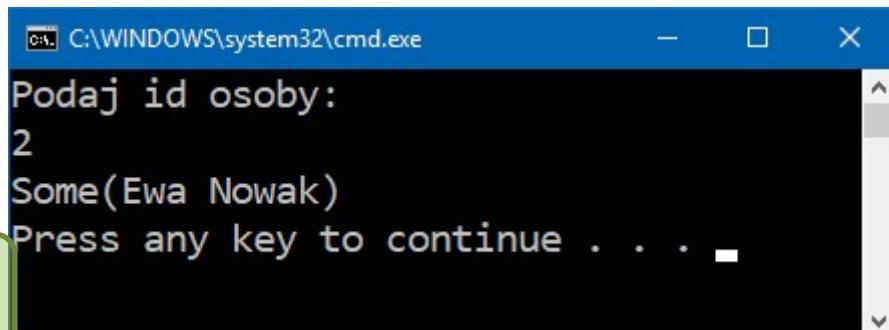
Podaj id osoby:
3
Some(Some(Tomek Kowalski))
Press any key to continue . . .

Mapowania

```
[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane"
    map (fun dane ->
        loop wczytajId
        |> znajdzOsobePoId dane
        |> map naString
    ) dane
    |> Option.flatten
    |> Console.WriteLine
0
'T option option → 'T option
```



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



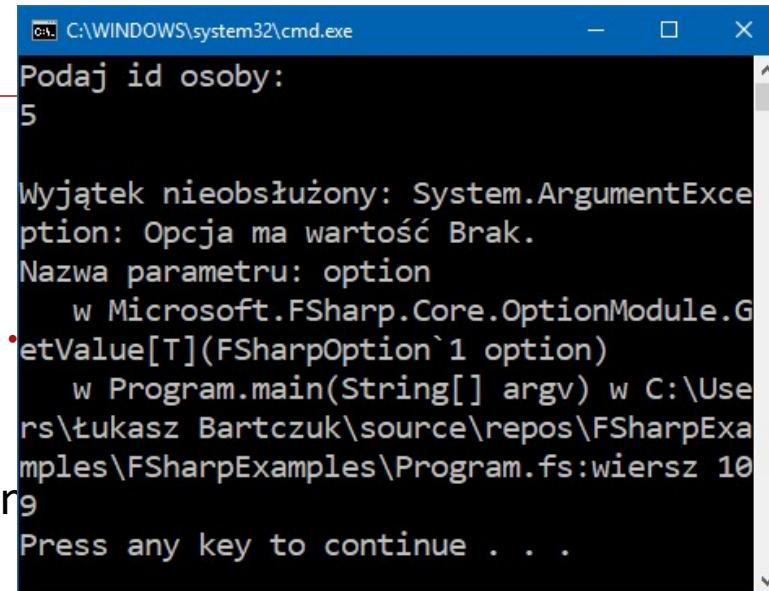
```
C:\WINDOWS\system32\cmd.exe
Podaj id osoby:
2
Some(Ewa Nowak)
Press any key to continue . . .
```

Mapowania

```
[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane."
    map (fun dane ->
        loop wczytajId
        |> znajdzOsobePoId dane
        |> map naString
    ) dane
    |> Option.flatten
    |> Option.get
    |> Console.WriteLine
```

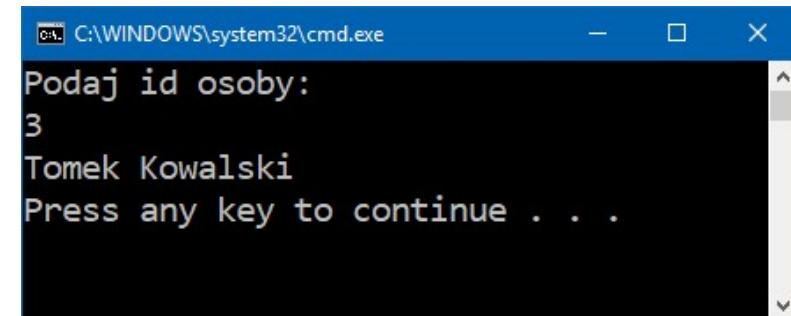
0

'T option → 'T



```
C:\WINDOWS\system32\cmd.exe
Podaj id osoby:
5

Wyjątek nieobsłużony: System.ArgumentException: Opcja ma wartość Brak.
Nazwa parametru: option
    w Microsoft.FSharp.Core.OptionModule.GetValue[T](FSharpOption`1 option)
    w Program.main(String[] argv) w C:\Users\Łukasz Bartczuk\source\repos\FSharpExamples\FSharpExamples\Program.fs:wiersz 10
Press any key to continue . . .
```

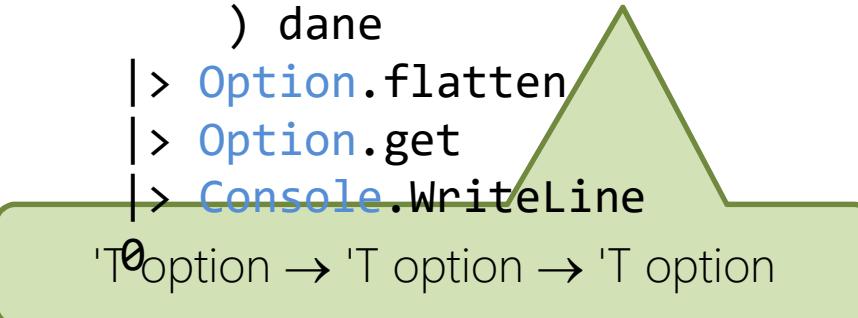


```
C:\WINDOWS\system32\cmd.exe
Podaj id osoby:
3
Tomek Kowalski
Press any key to continue . . .
```

Mapowania

```
[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane"
    map (fun dane ->
        loop wczytajId
        |> znajdzOsobePoId dane
        |> map naString
        |> Option.orElse (Some "Nie ma osoby o takim ID")
    ) dane
    |> Option.flatten
    |> Option.get
    |> Console.WriteLine
```

$'T^0 \rightarrow 'T \rightarrow 'T$



```
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

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

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

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

Monady

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

return: $(T) \rightarrow option < 'T >$

W programowaniu funkcyjnym dowolny typ wyposażony w funkcje Bind i Return nazywamy monadą

Bindowanie

```
[<EntryPoint>]
let main argv =
    let dane = wczytajDane "dane.json"
    bind (fun dane ->
        loop wczytajId
            |> znajdzOsobePoId dane
            |> map naString
            |> Option.orElse (Some "Nie ma osoby o takim ID")
        ) dane
    |> Option.get
    |> Console.WriteLine
    0
```

Bindowanie

```
let przetworz dane =
    loop wczytajId
    |> znajdzOsobePoId dane
    |> map naString
    |> Option.orElse (Some "Nie ma osoby o takim ID")
    |> Option.get
```

```
[<EntryPoint>]
let main argv =
    wczytajDane "dane.json"
    |> bind przetworz
    |> Console.WriteLine
    0
```

MONADA EITHER

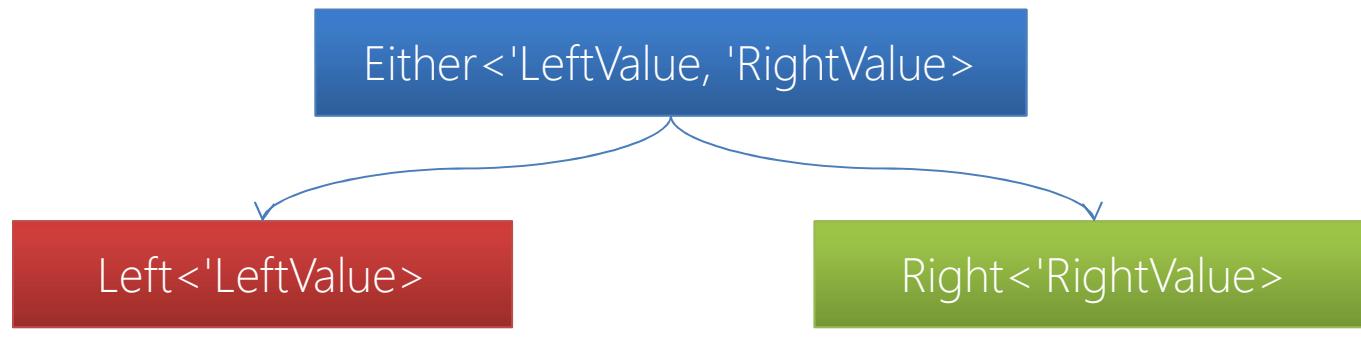
Either

Monada Either jest funkcyjną metodą pozwalającą na rozwiązywanie problemu obsługi błędów

Realizacja Railway Oriented Programming



Monada Either



Najczęściej przenosi
informacje o błędzie

Najczęściej przenosi
informacje poprawne

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

Monada Either

```
let map mapaPrawa opcje =
    match opcje with
    | Left x -> Left x
    | Right x -> Right (mapaPrawa x)
```

```
let bind mapaPrawa opcje =
    match opcje with
    | Left x -> Left x
    | Right x -> mapaPrawa x
```

Monada Either

```
let whenLeft akcja opcje =
  match opcje with
  | Left x -> akcja x
  | Right _ -> ()
  opcje
```

```
let whenRight akcja opcje =
  match opcje with
  | Left x -> ()
  | Right x -> akcja x
  opcje
```

Monada Either

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

Monada Either

```
let znajdzOsobePoId (lista:Osoba list) id =
    let poId (osoba:Osoba) = osoba.id = id

    if lista |> Seq.exists poId then
        Right (lista |> List.find poId)
    else
        Left "Nie ma osoby o takim id"
```

Monada Either

Monada Either

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

[<EntryPoint>]
let main argv =
    wczytajDane "dane.json"
    |> Either.bind przetworz
    |> Either.whenRight Console.WriteLine
    |> Either.whenLeft pokazBlad
    |> ignore
```

PROGRAMOWANIE GENERYCZNE

Funkcja zamien (C#)

```
...
void zamien(ref int a, ref int b)
{
    int tmp = a;
    a=b;
    b=tmp;
}
...
```

```
...
void zamien(ref long a, ref long b)
{
    long tmp = a;
    a=b;
    b=tmp;
}
...
```

```
...
void zamien(ref double a, ref double b)
{
    double tmp = a;
    a=b;
    b=tmp;
}
...
```

```
...
void zamien(ref float a, ref float b)
{
    float tmp = a;
    a=b;
    b=tmp;
}
...
```

Klasa Punkt (C#)

```
class Punkt
{
    private int _x, _y;

    public Punkt(int x,
                 int y) {...}
    public void SetX(int x) {...}
    public void SetY(int y) {...}
    public int GetX() {...}
    public int GetY() {...}
};
```

```
class Punkt
{
    private double _x, _y;

    public Punkt(double x,
                double y) {...}
    public void SetX(double x) {...}
    public void SetY(double y) {...}
    public double GetX() {...}
    public double GetY() {...}
};
```

Programowanie generyczne

Programowanie generyczne polega na tworzeniu algorytmów i struktur danych operujących na danych dowolnego typu (lub prawie dowolnego typu).

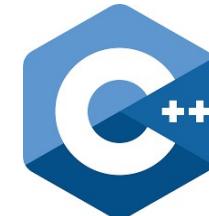
Paradygmat programowania generycznego jest najczęściej rozumiany jako uogólnienie paradymatu programowania obiektowego.

Programowanie generyczne w językach programowania

Generyki



Szablony



Klasa Punkt – wersja szablonowa

```
template <typename T> class Punkt
{
    private:
        T _x, _y;
    public:
        Punkt(T x, T y);
        void SetX(T x);
        void SetY(T y);
        T GetX();
        T GetY();
};

template <typename T> void Punkt<T>::SetX(T x)
{
    _x = x;
}
```

Funkcje generyczne w C#

```
class Klasa
{
    public static void zamien<T> (ref T a, ref T b)
    {
        T tmp = a;
        a=b;
        b=tmp;
    }
}
```

Klasy generyczne w C#

```
class Punkt<T> {
    private T _x, _y;

    public Punkt()
    {
        _x = default(T);
        _y = default(T);
    }
    public Punkt(T x, T y) {...}
    public void SetX(T x) {...}
    public void SetY(T y) {...}
    public T GetX() {...}
    public T GetY() {...}
};
```

Metody generyczne w C#

```
class Punkt<T> {
    private T _x, _y;

    public Punkt()
    {
        _x = default(T);
        _y = default(T);
    }
    public S Metoda<S>(T parametr) {...}
};
```

Ograniczenia generyczne (1)

```
class Stos<T>
{
    T[] bufor = new T[10];
    int gora = 0;

    public void Odloz(T wartosc) { bufor[gora++] = wartosc; }

    public T Zdejmij() {
        if (gora >= 1) {
            gora--; return bufor[gora];
        }
        else { return default (T); }
    }

    public bool Sprawdz(T x) {
        return bufor[gora - 1] == x;
    }
}
```

Ograniczenia generyczne (2)

```
class Stos<T> where T : IComparable<T>
{
    T[] bufor = new T[10];
    int gora = 0;

    public void Odloz(T wartosc) { bufor[gora++] = wartosc; }

    public T Zdejmij() {
        if (gora >= 1) {
            gora--; return bufor[gora];
        }
        else { return default (T); }
    }

    public bool Sprawdz(T x) {
        return bufor[gora - 1].CompareTo(x) == 0;
    }
}
```

Ograniczenia generyczne (2)

```
class Stos<T> where T : IComparable<T>, ICloneable
{
    T[] bufor = new T[10];
    int gora = 0;

    public void Odloz(T wartosc) { bufor[gora++] = wartosc; }

    public T Zdejmij() {
        if (gora >= 1) {
            gora--; return bufor[gora];
        }
        else { return default (T); }
    }

    public bool Sprawdz(T x) {
        return bufor[gora - 1].CompareTo(x);
    }
}
```

Ograniczenia generyczne (3)

```
class ListaUporzadkowana<K, V>
    where K : IComparable<K>
{
}
```

```
class ListaUporzadkowana<K, V>
    where K : IComparable<K>
    where V : ICloneable
{
}
```

Rodzaje ograniczeń generycznych

- ograniczenia dziedziczenia (parametr typu musi dziedziczyć po określonym typie)

```
class KlasaBazowa {}
class MojaKlasa<T> where T : KlasaBazowa {}
```

```
class MojaKlasa<T,U> where T : U { }
```

- ograniczenia konstruktora (parametr typu musi posiadać konstruktor domyślny)

```
class MojaKlasa<T> where T : new() {}
```

- ograniczenia typu

```
class MojaKlasa<T> where T : struct {}
```

```
class MojaKlasa<T> where T : class {}
```

Generyki C#, a szablony C++

- generyki C# nie są tak elastyczne, co szablony C++,
- C# nie pozwala na stworzenie szablonu dla konkretnego typu,
- C# nie pozwala na jawną specjalizację,
- C# nie pozwala na częściową specjalizację,
- C# nie pozwala aby parametr typu był klasą bazową dla typu generycznego,
- C# nie pozwala aby parametr typu miał domyślny typ,
- C# nie pozwala aby parametr typu był generyczny, chociaż typy tworzone mogą być użyte jako generyki,
- C# wymaga, aby cały kod był poprawny dla wszystkich typów jakie mogą być podstawione za parametry

Generyki - przykład

```
public class RepozytoriumOsob {  
    public List<Osoba> PobierzWszystkie() { ... }  
    public Osoba PobierzWgId(int id) { ... }  
    public void Dodaj(Osoba osoba) { ... }  
    public void Usun(Osoba osoba) { ... }  
    public void Edytuj(Osoba osoba) { ... }  
}
```

```
public class RepozytoriumKategorii {  
    public List<Kategoria> PobierzWszystkie() { ... }  
    public Kategoria PobierzWgId(int id) { ... }  
    public void Dodaj(Kategoria osoba) { ... }  
    public void Usun(Kategoria osoba) { ... }  
    public void Edytuj(Kategoria osoba) { ... }  
}
```

Generyki - przykład



Jak napisać funkcję, która będzie zwracała repozytorium na podstawie nazwy?

```
public class Fabryka {  
    public ... Stworz(string nazwa) { ... }  
}
```

Generyki - przykład

```
public interface IRepository<T> {  
    List<T> PobierzWszystkie();  
    T PobierzWgId(int id);  
    void Dodaj(T osoba);  
    void Usun(T osoba);  
    void Edytuj(T osoba);  
}
```

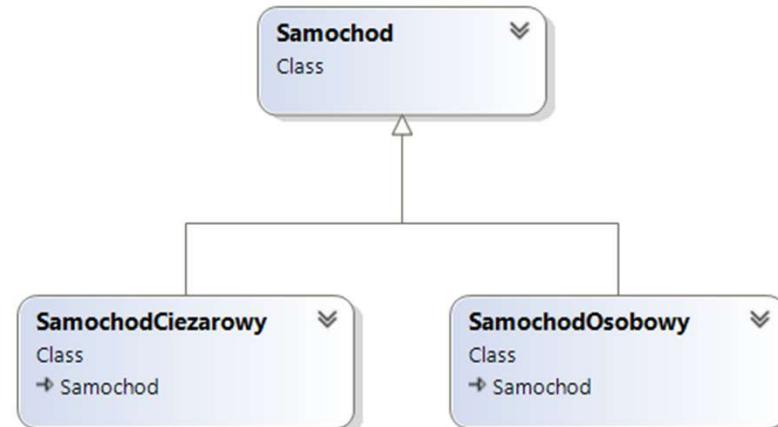
```
public class RepositoryOsob  
    : IRepository<Osoba> {  
    public List<Osoba> PobierzWszystkie() { ... }  
    public Osoba PobierzWgId(int id) { ... }  
    public void Dodaj(Osoba osoba) { ... }  
    public void Usun(Osoba osoba) { ... }  
    public void Edytuj(Osoba osoba) { ... }  
}
```

Kowariancja, kontrawariancja i inwariancja

Kowariancja pozwala na wykorzystanie bardziej szczegółowego typu niż pierwotnie określono

Kontrawariancja pozwala na wykorzystanie mniej szczegółowego typu niż pierwotnie określono

Inwariancja pozwala na wykorzystanie tylko takiego typu, który był oryginalnie określony.



```
class Samochod { }
```

```
class SamochodOsobowy : Samochod { }
```

```
class SamochodCiezarowy : Samochod { }
```

Dziedziczenie, a przypisanie





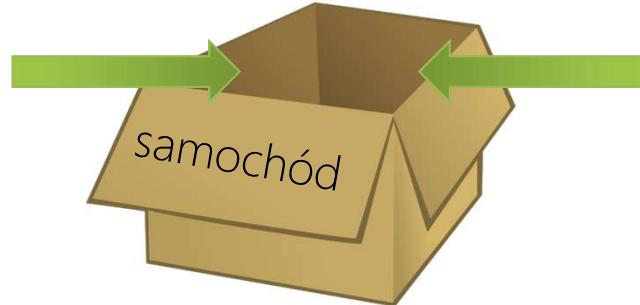
```
interface IFabryka<T>
{
    T Stworz();
}
```

```
class FabrykaFerrari : IFabryka<SamochodOsobowy> {
    public SamochodOsobowy Stworz() {
        throw new NotImplementedException();
    }
}
```

```
class FabrykaScania : IFabryka<SamochodCiezki> {
    public SamochodCiezki Stworz() {
        throw new NotImplementedException();
    }
}
```

```
class FabrykaVolvo : IFabryka<Samochod> {
    public Samochod Stworz() {
        throw new NotImplementedException();
    }
}
```

```
class SwiatWirutalny {  
  
    private Samochod samochod;  
  
    public void PotrzebnySamochod(IFabryka<Samochod> fabryka) {  
        samochod = fabryka.Stworz();  
    }  
}
```



```
class Program
{
    static void Main(string[] args)
    {
        SwiatWirutalny sw = new SwiatWirutalny();
        FabrykaFerrari ff = new FabrykaFerrari();
        FabrykaScania fs = new FabrykaScania();
        FabrykaVolvo fv = new FabrykaVolvo();
        ...
    }
}
```

```
sw.PotrzebnySamochod(ff);  
sw.PotrzebnySamochod(fs);  
sw.PotrzebnySamochod(fv);
```

Wywołanie poprawne

Wywołania błędne
Dlaczego?

Interfejs IFabryka<T> jest inwariantny

Inwariancja zwala na wykorzystanie tylko takiego typu, który był oryginalnie określony.

Aby nasz świat stał się idealny musimy interfejs przekształcić w kowariantny

```
interface IFabryka<out T>
{
    T Stworz();
}
```

Kowariancja pozwala na wykorzystanie bardziej szczegółowego typu niż pierwotnie określono



```
interface IService<T>
{
    void Napraw(T obiekt);
}
```

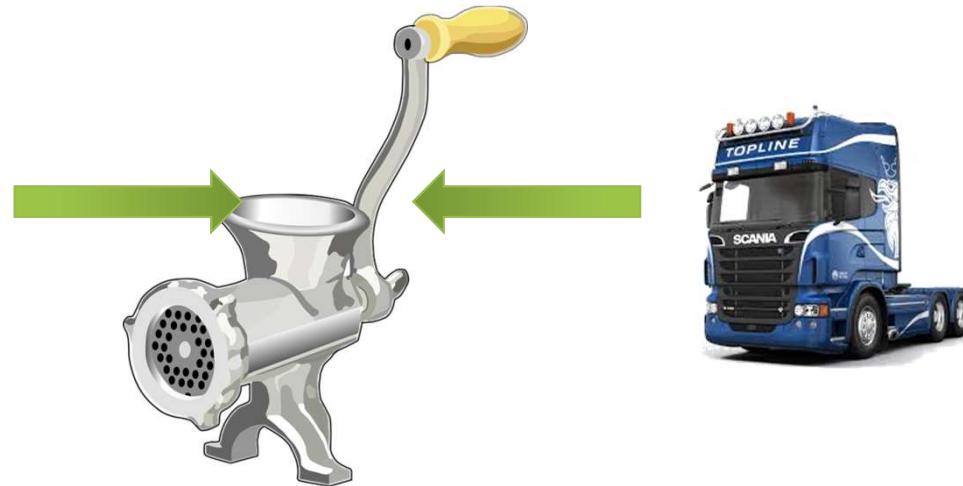
```
class ASOFerrari : ISerwis<SamochodOsobowy> {  
    public void Napraw(SamochodOsobowy obiekt) {  
        throw new NotImplementedException();  
    }  
}
```



```
class ASOScania : ISerwis<SamochodCiezarowy> {  
    public void Napraw(SamochodCiezarowy obiekt) {  
        throw new NotImplementedException();  
    }  
}
```



```
class SerwisGienka : ISerwis<Samochod> {  
    public void Napraw(Samochod obiekt) {  
        throw new NotImplementedException();  
    }  
}
```



```
class SamochodOsobowy : Samochod {  
    public void Napraw(ISerwis<SamochodOsobowy> serwis)  
    {  
        serwis.Napraw(this);  
    }  
}
```

```
class SamochodCiezarowy : Samochod {  
    public void Napraw(ISerwis<SamochodCiezarowy> serwis)  
    {  
        serwis.Napraw(this);  
    }  
}
```

```
SamochodOsobowy so = new SamochodOsobowy();  
  
so.Napraw(new ASOFerrari());  
so.Napraw(new ASOScania());  
so.Napraw(new SerwisGienka());
```

Wywołanie poprawne

Wywołania błędne
Dlaczego?

Interfejs ISerwis<T> jest invariantny

Aby nasz świat stał się idealny musimy interfejs przekształcić w kontrawariantny

```
interface ISerwis<in T>
{
    void Napraw(T obiekt);
}
```

WARTOŚCI NIEMODYFIKOWALNE W C#

Wartości modyfikowalne

```
class WartoscWZakresie {  
    public int Min { get; set; }  
    public int Max { get; set; }  
    public int Wartosc { get; set; }  
  
    public WartoscWZakresie(int min, int max, int wartosc) {  
        Min=min;  
        Max=max;  
        Wartosc = wartosc;  
    }  
  
    public void ZwiekszWartosc(int oWartosc) {  
        Wartosc += oWartosc;  
    }  
}
```

Wartości modyfikowalne

WartoscWZakresie w

```
= new WartoscWZakresie(min:-10, max:10, wartosc: 5);
```

WartoscWZakresie w1

```
= new WartoscWZakresie(10, 0, 5);
```

Obiekt jest całkowicie w błędnym stanie

Jeżeli mam obiekt to jest on poprawny

Wartości modyfikowalne

```
class WartoscWZakresie {  
    public int Min { get; set; }  
    public int Max { get; set; }  
    public int Wartosc { get; set; }  
  
    public WartoscWZakresie(int min, int max, int wartosc) {  
        Min=min;  
        Max=max;  
        Wartosc = wartosc;  
        SprawdzZakres();  
        SprawdzWartosc();  
    }  
    ...  
}
```

Wartości modyfikowalne

```
class WartoscWZakresie {  
    ...  
    private void SprawdzZakres() {  
        if(Min>Max)  
            throw new Exception();  
    }  
  
    private void SprawdzWartosc() {  
        if(wartosc<Min)  
            wartosc = Min;  
        if(wartosc>Max)  
            wartosc = Max;  
    }  
}
```

Wartości modyfikowalne

```
class WartoscWZakresie {  
  
    public int Min { get; private set; }  
    public int Max { get; private set; }  
    public int Wartosc { get; private set; }  
  
    void Zwiekszwartosc(int owartosc) {  
        Wartosc += owartosc;  
    }  
}
```

Wartości modyfikowalne

```
class WartoscWZakresie {  
  
    void ZwiekszWartosc(int oWartosc) {  
        Wartosc += oWartosc;  
        SprawdzWartosc();  
    }  
  
}
```

Wartości modyfikowalne

```
class WartoscWZakresie {  
  
    private int _min;  
  
    public int Min {  
        get { return _min; }  
        set {  
            _min = value;  
        }  
    }  
}
```

Wartości modyfikowalne

```
class WartoscWZakresie {  
  
    private int _min;  
    private int _max;  
  
    public int Min {  
        get { return _min; }  
        set {  
            if(value>_max)  
                throw new Exception();  
            _min = value;  
            SprawdzWartosc();  
        }  
    }  
}
```

Wartości niemodyfikowalne

Wartości niemodyfikowalne są to wartości, których składowe mogą być ustalone tylko w konstruktorze.

Wartości niemodyfikowalne

```
class WartoscWZakresie {  
  
    private readonly int _min;  
    private readonly int _max;  
    private readonly int _wartosc;  
  
    public int Min { get => _min; }  
    public int Max { get => _max; }  
    public int Wartosc { get => _wartosc; }  
  
    public WartoscWZakresie(int min, int max, int wartosc) {  
        _min=min;  
        _max=max;  
        _wartosc = wartosc;  
        SprawdzZakres();  
        SprawdzWartosc();  
    }  
    ...
```

Wartości niemodyfikowalne

```
class WartoscWZakresie {  
    ...  
    public WartoscWZakresie ZwiekszWartosc(int oWartosc) {  
        return new WartoscWZakresie(_min, _max, _wartosc+oWartosc);  
    }  
    public WartoscWZakresie ZmienMin(int min) {  
        return new WartoscWZakresie(min, _max, _wartosc);  
    }  
}
```

FUNKCJE WYŻSZYCH RZĘDÓW

Filtrowanie listy

Problem: przefiltrować listę



```
class Program
{
    static void Main(string[] args)
    {
        List<string> przedmioty = new List<string>() {
            "Paradygmaty programowania",
            "Programowanie stron internetowych",
            "Algebra",
            "Logika dla informatyków",
            "Sieci komputerowe"
        };
    }
}
```

Filtrowanie listy

```
class OperacjeNaLiscie {
    public static List<string> TylkoDlugieNazwy(List<string> lista) {
        List<string> rezultat = new List<string>();
        for(int i=0; i<lista.Count; i++)
            if(lista[i].Length > 18)
                rezultat.Add(lista[i]);
        return rezultat;
    }

    public static List<string> TylkoNaLitereP(List<string> lista) {
        List<string> rezultat = new List<string>();
        for(int i=0; i<lista.Count; i++)
            if(lista[i][0] == 'P')
                rezultat.Add(lista[i]);
        return rezultat;
    }
}
```

Filtrowanie listy

```
class Program
{
    static void Main(string[] args)
    {
        List<string> przedmioty = new List<string>() {
            "Paradygmaty programowania",
            ...
        };

        List<string> dlugieNazwy
            = OperacjeNaLiscie.TylkoDlugieNazwy(przedmioty);
        List<string> przedmiotyNaLitereP
            = OperacjeNaLiscie.TylkoNaLitereP(przedmioty);
    }
}
```

Filtrowanie listy



Czy można to rozwiązać
bardziej elegancko?

Filtrowanie listy

```
class OperacjeNaLiscie {
    public static List<string> TylkoDlugieNazwy(List<string> lista) {
        List<string> rezultat = new List<string>();
        for(int i=0; i<lista.Count; i++)
            if(lista[i].Length > 18)
                rezultat.Add(lista[i]);
        return rezultat;
    }

    public static List<string> TylkoNaLitereP(List<string> lista) {
        List<string> rezultat = new List<string>();
        for(int i=0; i<lista.Count; i++)
            if(lista[i][0] == 'P')
                rezultat.Add(lista[i]);
        return rezultat;
    }
}
```



NYFell.com

Wzorzec Strategii

Czynnościowy wzorzec projektowy, który definiuje rodzinę wymiennych algorytmów i kapsułkuje je w postaci klas.

Wzorzec strategii

```
abstract class Warunek {  
    public abstract bool Sprawdz(string wartosc);  
}
```

```
abstract class Warunek <Twartosc> {  
    public abstract bool Sprawdz(Twartosc wartosc);  
}
```

Wzorzec strategii

```
class TylkoDlugieNazwy : Warunek<string> {

    public override bool Sprawdz(string wartosc) {
        return wartosc.Length > 18;
    }

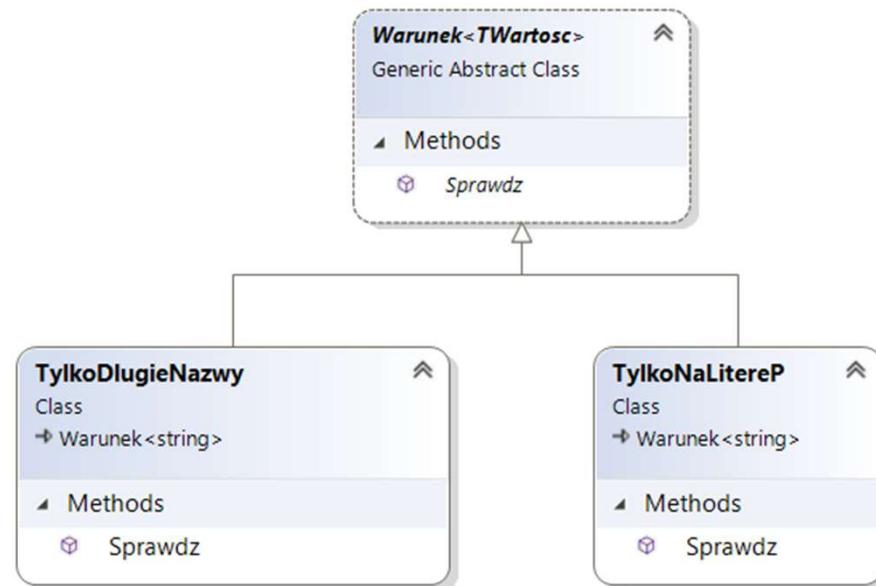
}

class TylkoNaLitereP : Warunek<string> {

    public override bool Sprawdz(string wartosc) {
        return wartosc[0] == 'P';
    }

}
```

Wzorzec strategii



Wzorzec strategii

```
class OperacjeNaLiscie {
    public static List<string> Filtruj(
        List<string> lista,
        Warunek<string> warunek)
    {
        List<string> rezultat = new List<T>();
        for(int i=0; i<lista.Count; i++)
            if(warunek.Sprawdz(lista[i]))
                rezultat.Add(lista[i]);
        return rezultat;
    }
}
```

Wzorzec strategii

```
class OperacjeNaLiscie {
    public static List<T> Filtruj<T>(
        List<T> lista,
        Warunek<T> warunek)
    {
        List<T> rezultat = new List<T>();
        for(int i=0; i<lista.Count; i++)
            if(warunek.Sprawdz(lista[i]))
                rezultat.Add(lista[i]);
        return rezultat;
    }
}
```

Wzorzec strategii

```
class Program {
    static void Main(string[] args) {
        List<string> przedmioty = new List<string>() {
            "Paradygmaty programowania",
            ...
        };

        List<string> dlugieNazwy
            = OperacjeNaLiscie.Filtruj(przedmioty, new TylkoDlugieNazwy());
        List<string> przedmiotyNaLitereP
            = OperacjeNaLiscie.Filtruj(przedmioty, new TylkoNaLitereP());
    }
}
```

Funkcje wyższych rzędów

Funkcje wyższych rzędów są to funkcje, które przyjmują inne funkcje jako parametry, lub funkcje zwracają.

Funkcje wyższych rzędów

```
class OperacjeNaLiscie {
    public static List<T> Filtruj<T>(
        List<T> lista,
        Func<T, bool> warunek)
    {
        List<T> rezultat = new List<T>();
        for(int i=0; i<lista.Count; i++)
            if(warunek(lista[i]))
                rezultat.Add(lista[i]);
        return rezultat;
    }
}
```

Funkcje wyższych rzędów

```
class Program {  
  
    static bool TylkoDlugie(string wartosc) => wartosc.Length > 18;  
    static bool TylkoNaP(string wartosc) => wartosc[0] == 'P';  
    static void Main(string[] args) {  
  
        List<string> przedmioty = new List<string>() {  
            "Paradygmaty programowania",  
            ...  
        };  
  
        List<string> dlugieNazwy  
            = OperacjeNaLiscie.Filtruj(przedmioty, TylkoDlugie);  
        List<string> przedmiotyNaLitereP  
            = OperacjeNaLiscie.Filtruj(przedmioty, TylkoNaP);  
    }  
}
```

Funkcje wyższych rzędów

```
class Program {  
  
    static bool TylkoDlugie(string wartosc) => wartosc.Length > 18;  
  
    static void Main(string[] args) {  
  
        List<string> przedmioty = new List<string>() {  
            "Paradygmaty programowania",  
            ...  
        };  
  
        List<string> dlugieNazwy  
        = OperacjeNaLiscie.Filtruj(przedmioty, (nazwa)=>nazwa.Length>18);  
    }  
}
```

Wzorzec strategii

```
class TylkoNaLitere : Warunek<string> {

    char _litera;
    public TylkoNaLitere(char litera) {
        _litera = litera;
    }

    public bool Sprawdz(string wartosc) {
        return wartosc[0] == _litera;
    }

}
```

Funkcje wyższych rzędów

```
class Program {

    static bool TylkoNaLitere(string nazwa, char litera) => nazwa[0] == litera;

    static Func<string, bool> TylkoNaLitere(char litera) {
        return (nazwa) => TylkoNaLitere(nazwa, litera);
    }

    static void Main(string[] args)  {

        List<string> przedmioty = new List<string>() {
            "Paradygmaty programowania", ...
        };

        List<string> dlugieNazwy
            = OperacjeNaLiscie.Filtruj(przedmioty, TylkoNaLitere('S'));
    }
}
```

Funkcje wyższych rzędów

```
class Program {

    static Func<string, bool> TylkoNaLitere(char litera) {
        return (nazwa) => nazwa[0] == litera;
    }

    static void Main(string[] args)  {

        List<string> przedmioty = new List<string>() {
            "Paradygmaty programowania", ...
        };

        List<string> dlugieNazwy
            = OperacjeNaLiscie.Filtruj(przedmioty, TylkoNaLitere('S'));
    }
}
```

OBLICZENIA LENIWE

Zachłanne i leniwe obliczenia

Obliczenia zachłanne wymagają wyznaczenia wartości parametrów aktualnych przed wywołaniem funkcji, obliczenia leniwe nie.

```
double X(bool a, double b, double  
{  
    if (a)  
        return b + 2;  
    else  
        return c + 2;  
}
```

Tutaj będzie obliczone
`Math.Sqrt(-1.0)`

W obliczeniach zachłannych to
wywołanie będzie błędne

`X(true, 12, Math.Sqrt(-1.0))`

W obliczeniach leniwych ten kod
się wykona

Leniwe obliczenia

Implementacja za pomocą Func<>

```
public double X(Func<bool> a, Func<int> b, Func<double> c)
{
    return a() ? b() + 1 : c() + 1;
}

X(() => true, () => 12, () => Math.Sqrt(-1.0));
```

Leniwe obliczenia

Implementacja za pomocą Lazy<>

```
public double X(Lazy<bool> a, Lazy<int> b, Lazy<double> c)
{
    return a() ? b() + 1 : c() + 1;
}

X(new Lazy<bool>(() => true),
  new Lazy<int>(() => 12),
  new Lazy<double>(() => Math.Sqrt(-1.0)));
```

Leniwe sekwencje

Sekwencje, w których kolejne elementy nie są obliczane natychmiast, ale w momencie gdy są potrzebne.

IEnumerable<T>, IEnumerator<T>

```
public interface IEnumerator
{
    object Current { get; }
    bool MoveNext();
    void Reset();
}

public interface IEnumerator<out T> : IDisposable, IEnumerator
{
    T Current { get; }
}
```

IEnumerable<T>, IEnumerator<T>

```
public interface IEnumerable<out T> : IEnumerable
{
    IEnumerator<T> GetEnumerator();
}
```

Fibonacci

```
class Fibonacci : IEnumarator<Tuple<int, int>>, IEnumberable<Tuple<int, int>>
{
    private int f_1 = 1;
    private int f_2 = 0;
    private int element = -1;

    public Tuple<int, int> Current => Tuple.Create(f_1, element);

    object IEnumberator.Current => Tuple.Create(f_1, element);

    public bool MoveNext() {
        var tmp = f_1;
        f_1 = f_2;
        f_2 = tmp + f_1;
        element++;
        return true;
    }
}
```

Fibonacci

```
public void Reset() {
    f_1 = 1;
    f_2 = 0;
    element = -1;
}

public void Dispose() { }

public IEnumarator<Tuple<int, int>> GetEnumerator() => this;

IEnumarator IEnumerable.GetEnumerator() => this;
}
```

Fibonacci

```
var fib = new Fibonacci();

foreach (var f in fib) {
    Console.WriteLine($"{f.Item2} {f.Item1}");
    if (f.Item1 > 10)
        break;
}
```

Funkcje iteracyjne

```
public static IEnumerable<int> Example()
{
    yield return 1;
    yield return 2;
    yield return 3;
}
```

Funkcje iteracyjne

```
public static IEnumerable<Tuple<int,int>> Fib() {
    int f_1 = 0;
    int f_2 = 1;
    int element = 0;

    do {
        int tmp = f_1;
        f_1 = f_2;
        f_2 = tmp + f_2;
        yield return Tuple.Create(tmp, element++);
    }
    while (true);
}
```

Funkcje iteracyjne

```
private sealed class <Example>d_3 :  
    IEnumerable<int>, IEnumerable, IEnumerator<int>,  
    IDisposable, IEnumerator  
{  
    private int <>1__state;  
    private int <>2__current;  
    private int <>1__initialThreadId;  
  
    public <Example>d_3(int _param1) {  
        base..ctor();  
        this.<>1__state = _param1;  
        this.<>1__initialThreadId = Environment.CurrentManagedThreadId;  
    }  
}
```

Funkcje iteracyjne

```
bool IEnumerator.MoveNext() {
    switch (this.<>1__state) {
        case 0:
            this.<>1__state = -1; this.<>2__current = 1;
            this.<>1__state = 1; return true;
        case 1:
            this.<>1__state = -1; this.<>2__current = 2;
            this.<>1__state = 2; return true;
        case 2:
            this.<>1__state = -1; this.<>2__current = 3;
            this.<>1__state = 3; return true;
        case 3:
            this.<>1__state = -1; return false;
        default:
            return false;
    }
}
```

Funkcje iteracyjne

```
void IDisposable.Dispose() {}

int IEnumerator<int>.Current { get { return this.<>2__current; } }

void IEnumerator.Reset() { throw new NotSupportedException(); }

object IEnumerator.Current { get { return (object) this.<>2__current; } }
```

METODY ROZSZERZAJĄCE

Problem

Przygotowanie metody, która pozwala z łańcucha znaków wybrać co drugi znak.

Podejście 1 – modyfikacja klasy string

```
class String {  
    public string CoDrugi() {  
        var sb = new StringBuilder();  
        for(int i=0; i<Length; i++)  
            if(i%2 == 0)  
                sb.Append(this[i]);  
        }  
        return sb.ToString();  
    }  
}
```



Brak możliwości zmodyfikowania klasy String.

Podejście 2 – tworzenie klasy pochodnej

```
class MyString : String {  
    public string CoDrugi() {  
        var sb = new StringBuilder();  
        for(int i=0; i<Length; i++)  
            if(i%2 == 0)  
                sb.Append(this[i]);  
        }  
        return sb.ToString();  
    }  
}
```



Podejście 3 – tworzenie nowej klasy statycznej

```
class static MyString {  
    public static string CoDrugi(string text) {  
        var sb = new StringBuilder();  
        for(int i=0; i< text.Length; i++) {  
            if(i%2 == 0) sb.Append(text[i]);  
        }  
        return sb.ToString();  
    }  
}  
  
string text = "Ala ma  
kota";  
MyString.CoDrugi(text);
```

Metody rozszerzające

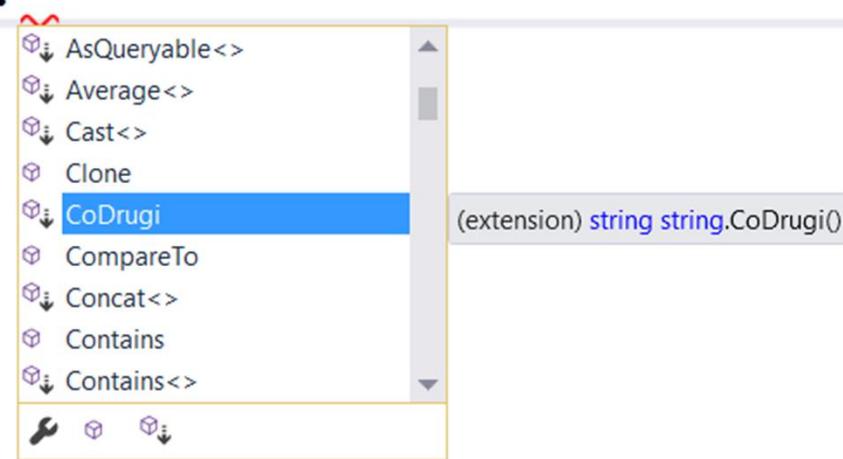
Metody rozszerzające pozwalają na dodanie nowych metod do już istniejących typów danych bez konieczności tworzenia nowego typu lub modyfikowania istniejącego kodu.

Podejście 4 – metody rozszerzające

Nazwa klasy jest bez
znaczenia

```
static class Extensions {  
    public static string CoDrugi(this string text) {  
        var sb = new StringBuilder();  
        for(int i=0; i< text.Length; i++) {  
            if(i%2 == 0) sb.Append(text[i]);  
        }  
        return sb.ToString();  
    }  
}  
  
string text = "Ala ma  
kota";  
text.CoDrugi();
```

```
public void Metoda()
{
    string text = "Ala ma kota";
    string rezultat = text.
```



PROGRAMOWANIE FUNKCYJNE W C# - LINQ

LINQ – Wstęp

LINQ – Language Integrated Query

Technologia LINQ pozwala na ujednolicony sposób odpytywania dowolnej kolekcji danych.

Składnia LINQ wzorowana jest na języku SQL.

Linq to Objects

LINQ to Objects służy do odpytywania kolekcji obiektów takich jak:

- listy
- tablice
- kolejki
- ...

Zapytania

```
var rezultat =
```

```
    (from v in listaLosowa
```

```
        where v > 10
```

```
        select v * v
```

```
).ToList();
```

Kolekcja źródłowa

Warunek wyboru

Mapowanie

Operacje mapowania elementów

Operacja mapowania listy tworzy nową listę poprzez zastosowanie funkcji f do każdego elementu listy wejściowej

$\text{map}: (A[], A \rightarrow B) \rightarrow B[]$

$\text{map}(\boxed{a1 \quad a2 \quad a3 \quad a4 \quad a5 \quad a6 \quad a7 \quad a8 \quad a9}, f)$



$\boxed{f(a1) \quad f(a2) \quad f(a3) \quad f(a4) \quad f(a5) \quad f(a6) \quad f(a7) \quad f(a8) \quad f(a9)}$

Operacje mapowania elementów

```
public static IEnumerable<TResult> Select<TSource, TResult>
    (this IEnumerable<TSource> source,
     Func<TSource, TResult> selector)
```

```
var lista = (from v in Enumerable.Range(0, 100)
            select rand.Next(0, 99))
            .ToList();
```

```
var kwadraty = (from v in lista
                  select v * v
                  ).ToArray();
```

Operacje mapowania elementów

```
List<Punkt> punkty = new List<Punkt> {  
    new Punkt { Kolor = Color.Red, X=1, Y=1 },  
    new Punkt { Kolor = Color.Blue, X=10, Y=10 },  
    new Punkt { Kolor = Color.Green, X=-10, Y=-10 }  
};
```

[?] (local variable) List<'a> wybrane

Anonymous Types:

'a is new { int X, int DW }

```
var wybrane = (from p in punkty  
               select new { p.X, DW = p.Y })  
.ToList();
```

Operacja filtrowania listy

Operacja filtrowania listy tworzy nową listę poprzez wybranie tych elementów listy, które spełniają określony warunek

$filtr: (A[], A \rightarrow \text{bool}) \rightarrow A[]$

$filtr([a1 \ a2 \ a3 \ a4 \ a5 \ a6 \ a7 \ a8 \ a9] , p)$



$a1 \ a3 \ a4 \ a7 \ a8 \ a9$

Operacja filtrowania listy

```
public static IEnumerable<TSource> Where<TSource>(  
    this IEnumerable<TSource> source,  
    Func<TSource, bool> predicate);
```

```
var parzyste = (from v in lista  
                where v % 2 == 0  
                select v  
            );
```

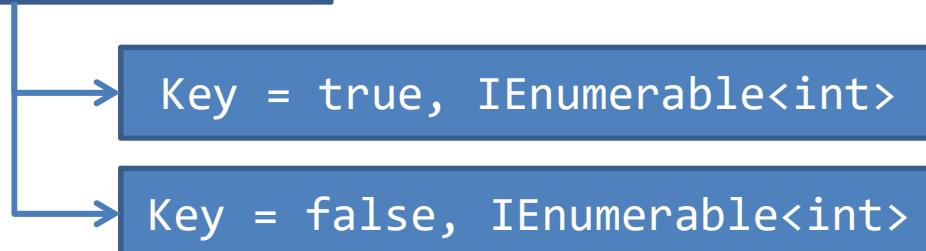
Operacja grupowania

Operacja grupowania pozwala podzielić kolekcje na podzbiory

```
var grupy = (from v in lista  
             group v by v % 2 == 0  
           );
```



IEnumerable<IGrouping>



Złączenia

```
var join = (from v1 in lista
            join v2 in kwadraty on v1 equals v2
            select v1
        ).ToList();
```

Zmienne zakresu i let

Zmienne zakresu reprezentują każdy element sekwencji w kolejności

```
var wybrane = (from ksiazka in ksiazki  
                let pLitera = ksiazka.Tytul[0] == 'P'  
                let lStron = ksiazka.Strony > 250  
                where pLitera && lStron  
                select ksiazka  
            ).ToList();
```

Sortowanie list



```
var listaLosowa  
= new List<int>();  
...  
listaLosowa.Sort();
```

```
var listaLosowa  
= new List<int>();  
...  
listaLosowa.OrderBy(v=>v)  
.ToList();
```

Odroczone wykonanie

```
IEnumerable<int> numbers =  
    Enumerable.Range(0, 10)  
        .Select(n => n * n);
```

Nie spowoduje
wykonania zapytania

- `ToList`
- `ToArray`
- `ToDictionary`
- `ToLookup`

Operacja agregowania elementów (1)

Operacja agregowania tworzy pojedynczą wartość z połączenia elementów listy

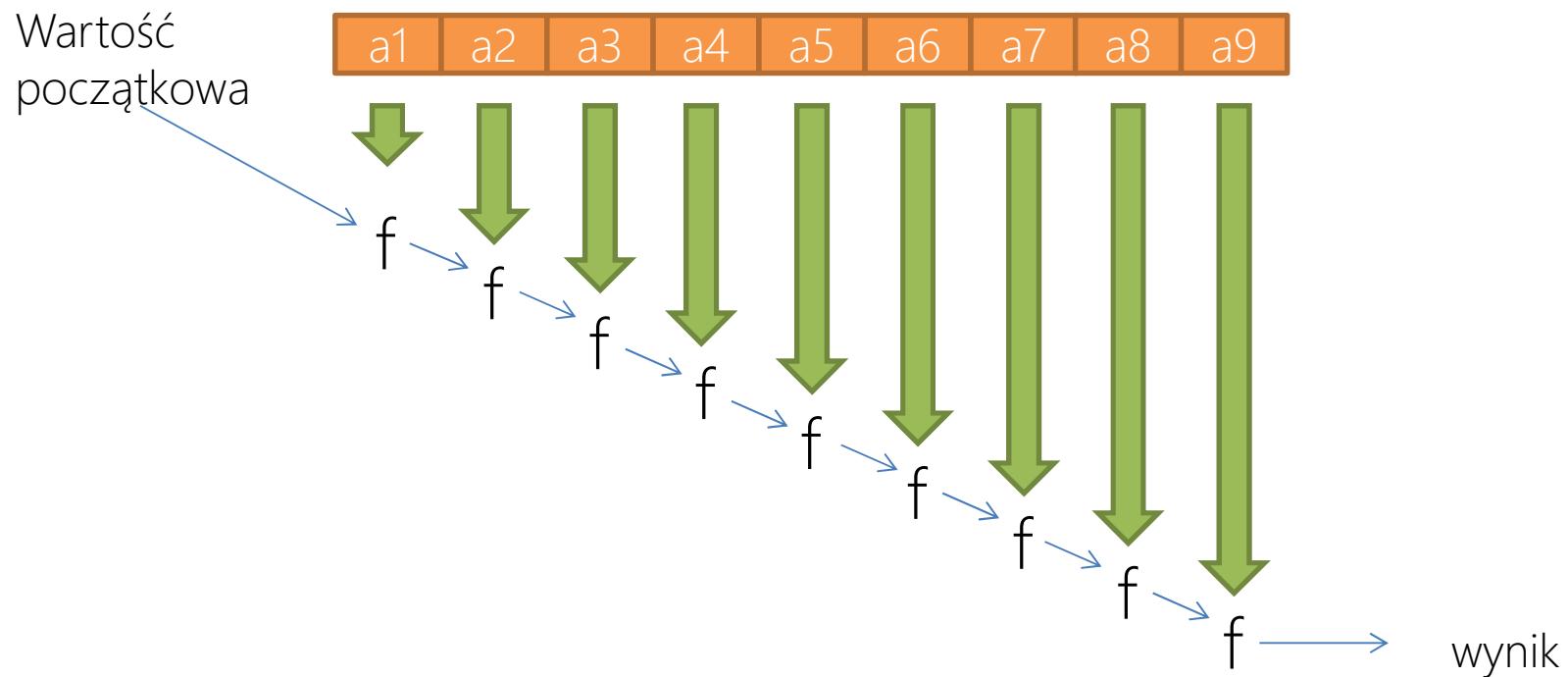
$\text{agreguj}(A[], R, (R, A) \rightarrow R) \Rightarrow R$



Wartość początkowa

Funkcja agregująca

Operacja agregowania elementów (2)



Operacja agregowania elementów (3)

```
int sum = Enumerable.Range(0, 10)
    .Aggregate(0, (a, v)=>a+v);
```

```
listaLosowa
    .Aggregate(listaLosowa[0],
        (a, v) => a > v ? v : a)
```

Operacja agregowania elementów (4)

```
int sum = Enumerable.Range(0, 10)
    .Aggregate(0, (a, v)=>a+v);
```

```
listaLosowa
    .Aggregate(listaLosowa[0],
        (a, v) => a > v ? v : a)
```

Zapytania

```
var rezultat = listaLosowa
    .Where(v => v > 10)
    .Select(v => v * v).ToList();
```

```
var rezultat =
    (from v in listaLosowa
     where v > 10
     select v * v
    ).ToList();
```

Odroczone wykonanie

```
IEnumerable<int> numbers =  
    Enumerable.Range(0, 10)  
        .Select(n => n * n);
```

Nie spowoduje
wykonania zapytania

- `ToList`
- `ToArray`
- `ToDictionary`
- `ToLookup`

Linq to Objects - operatory

Operator	Grupa	Operator	Grupa
Aggregate	Agregacja	ElementAt	Elementy
All	Liczебность	ElementAtOrDefault	Elementy
Any	Liczебность	Empty	Tworzenie
AsEnumerable	Konwersja	Except	Zbiór
Average	Agregacja	First	Elementy
Cast	Konwersja	FirstOrDefault	Elementy
Concat	Zbiór	GroupBy	Grupowanie
Contains	Liczебность	GroupJoin	Połączenia
Count	Agregacja	Intersect	Zbiór
DefaultIfEmpty	Elementy	Join	Elementy
Distinct	Zbiór	Last	Elementy

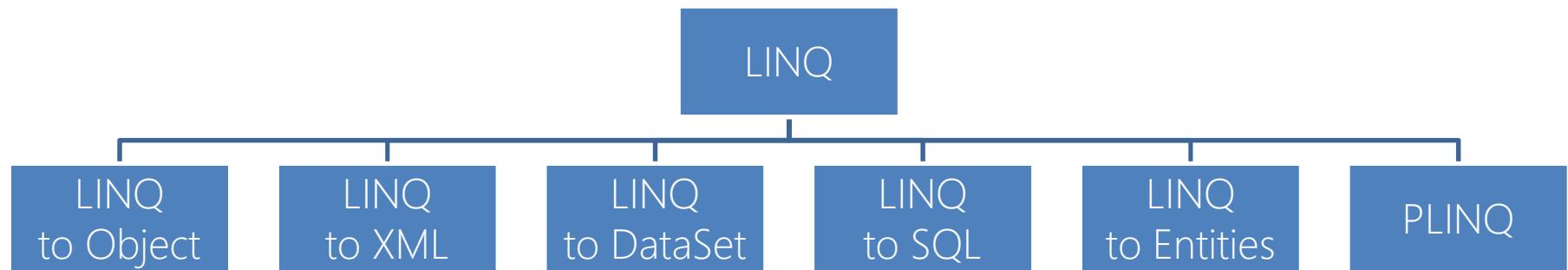
Linq to Objects - operatory

Operator	Grupa	Operator	Grupa
LastOrDefault	Elementy	SelectMany	Projekcja
LongCount	Agregacja	SequenceEqual	Równość
Max	Agregacja	Single	Elementy
Min	Agregacja	SingleOrDefault	Elementy
OfType	Konwersja	Skip	Podział
OrderBy	Porządkowanie	SkipWhile	Podział
OrderByDescending	Porządkowanie	Sum	Agregacja
Range	Generacja	Take	Podział
Repeat	Generacja	TakeWhile	Podział
Reverse	Porządkowanie	ThenBy	Porządkowanie
Select	Projekcja	ThenByDescending	Porządkowanie

LINQ to Objects - operatory

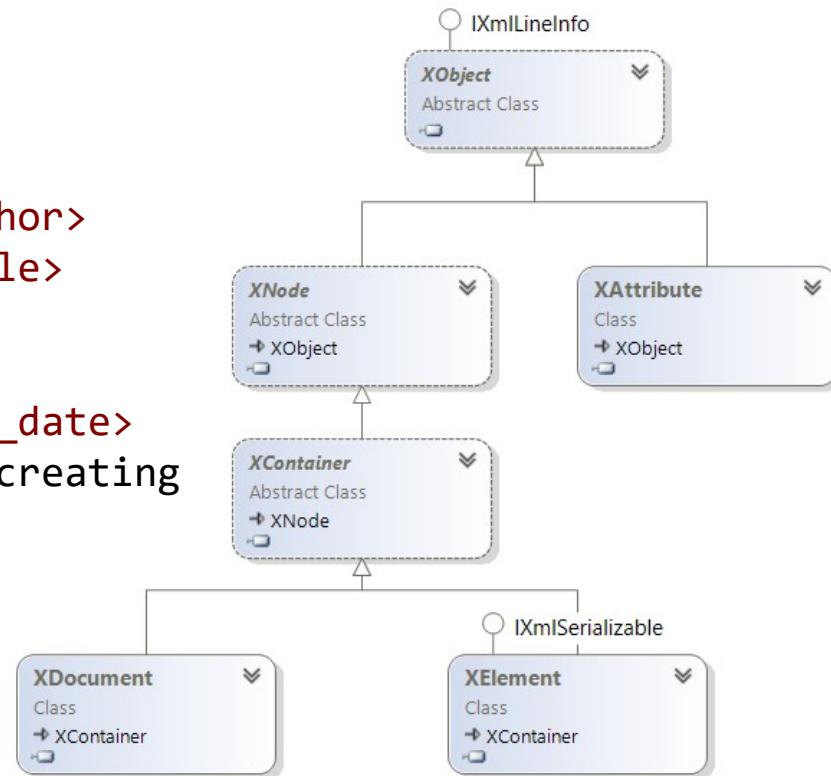
Operator	Grupa	Operator	Grupa
ToArray	Konwersja	ToLookup	Konwersja
ToDictionary	Konwersja	Union	Zbiór
ToList	Konwersja	Where	Ograniczenia

Linq



Linq To XML

```
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating
      applications with XML.
    </description>
  </book>
</catalog>
```



Tworzenie elementów (1)

```
var przyklad =
    @"<catalog>
        <book id='bk101'>
            <author>Gambardella, Matthew</author>
            <title>XML Developer's Guide</title>
            <genre>Computer</genre>
            <price>44.95</price>
            <publish_date>2000-10-01</publish_date>
            <description>
                An in-depth look at creating applications
                with XML.
            </description>
        </book>
    </catalog>";
var katalog = XDocument.Parse(przyklad);
```

Tworzenie elementów (2)

```
var katalog = new XElement("catalog");
var ksiazka = new XElement("book");
    ksiazka.Add(new XAttribute("id", "bk101"));
    ksiazka.Add(new XElement("author", "Gambardella, Matthew"));
    ksiazka.Add(new XElement("title", "XML Developer's Guide"));
katalog.Add(ksiazka);
```

Tworzenie elementów (3)

```
var katalog =  
    new XElement("catalog",  
        new XElement("book",  
            new XAttribute("id", "bk101"),  
            new XElement("author", "Gambardella, Matthew"),  
            new XElement("title", "XML Developer's Guide")  
        )  
    );
```

Wyszukiwanie elementów (1)

Przeszukiwanie dzieci	Przeszukiwanie rodziców
Metody / właściwości	Metody / właściwości
FirstNode	Parent
LastNode	Ancestors() / Ancestors(Name)
Element(Name)	
Elements() / Elements(Name)	Wartość węzła
Nodes()	Value
DescendantNodes()	
Descendants() / Descendants(Name)	

Wyszukiwanie elementów (2)

```
var tytulyKsiazek = katalog
    .Descendants("title")
    .Select(w=>w.Value)
    .ToList();
```

```
var tytulyKsiazek =
(
    from wezel in doc.Descendants("title")
    select wezel.Value
).ToList();
```

Wyszukiwanie elementów (3)

```
var listaKsiazek =
    katalog
        .Descendants("book")
        .Where(d => d.Element("author").Value.Contains("Corets"))
        .ToList();

var listaKsiazek =
    ( from ksiazka in doc.Descendants("book")
      where ksiazka.Element("author")
            .Value
            .Contains("Corets")
      select ksiazka
    ).ToList();
```

SelectMany (1)

Operator SelectMany pozwala spłaszczyć hierarchię sekwencji

```
var przedmioty = new List<string> {
    "Algebra",
    "Paradygmaty programowania",
    "Systemy Operacyjne"
};

var litery = przedmioty
    .SelectMany(p => p.ToCharArray())
    .ToList();
```

SelectMany (2)

```
var litery = przedmioty
    .SelectMany(p =>p.ToCharArray())
    .ToList();
```

```
static IEnumerable<R> SelectMany<A, R>(
    this IEnumerable<A> sequence,
    Func<A, IEnumerable<R>> function)
{
    foreach (A outerItem in sequence)
        foreach (R innerItem in function(outerItem))
            yield return innerItem;
}
```

SelectMany (3)

```
var litery = przedmioty  
    .SelectMany(p =>p.ToCharArray())  
    .ToList();
```

```
var litery = (  
    from przedmiot in przedmioty  
    from litera in przedmiot.ToCharArray()  
    select litera  
).ToList();
```

SelectMany (4)

```
var a = new[] { 1, 4, 7 };
var b = new[] { 2, 5, 8 };
var wynik = a.SelectMany(i => b, (i, j) => i + j).ToList();
```

```
public static IEnumerable<C> SelectMany<A, B, C>(
    this IEnumerable<A> items,
    Func<A, IEnumerable<B>> function,
    Func<A, B, C> projection)
{
    foreach (A outer in items)
        foreach (B inner in function(outer))
            yield return projection(outer, inner);
}
```

SelectMany (5)

```
var a = new[] { 1, 4, 7 };
var b = new[] { 2, 5, 8 };
var wynik = a.SelectMany(i => b, (i, j) => i + j).ToList();
```

```
var wynik = (
    from v1 in a
    from v2 in b
    select v1 + v2
).ToList();
```

SelectMany (6)

```
<plan_grup>
    <dzientyg>Poniedzia³ek</dzientyg>
    <godz>8.00 - 9.00</godz>
    <oddz>Informatyka I-go st sem.4 gr. dziekañska 1 lab.1</oddz>
    <naucz>Dr hab. inż. prof. PCz /IISI/</naucz>
    <imie>Cpa³ka Krzysztof </imie>
    <przedm>Programowanie obiektowe wyk.</przedm>
    <sala>s. A-4</sala>

    <dzientyg>Poniedzia³ek</dzientyg>
    <godz>8.00 - 9.00</godz>
    <oddz>Informatyka I-go st sem.4 gr. dziekañska 1 lab.2</oddz>
    <naucz>Dr hab. inż. prof. PCz /IISI/</naucz>
    <imie>Cpa³ka Krzysztof </imie>
    <przedm>Programowanie obiektowe wyk.</przedm>
    <sala>s. A-4</sala>
</plan_grup>
```

SelectMany (7)

```
var pon = plan.Descendants("dzientyg")
    .Where(dzien => dzien.Value == "Poniedzia³ek")
    .Select(dzien => dzien.ElementsAfterSelf().Take(16));

var ponI509 = pon
    .Where(dzien => dzien.Any(n => n.Value.Contains("509")));

var godziny = ponI509
    .SelectMany(wezly => wezly.Where(
        wezel => wezel.Name == "godz"),
        (wezly, godzina) => godzina.Value)
    .ToList();
```

SelectMany (8)

```
var pon = (from dzien in plan.Descendants("dzientyg")
           where dzien.Value == "PoniedziaŁek"
           select dzien.ElementsAfterSelf().Take(16)
         );

var ponI509 = (from jedn in pon
               where jedn.Any(n => n.Value.Contains("509"))
               select jedn
             );

var godziny = (from jedn in ponI509
               from godzina in
                   jedn.Where(wezel => wezel.Name == "godz")
               select godzina.Value
             ).ToList();
```

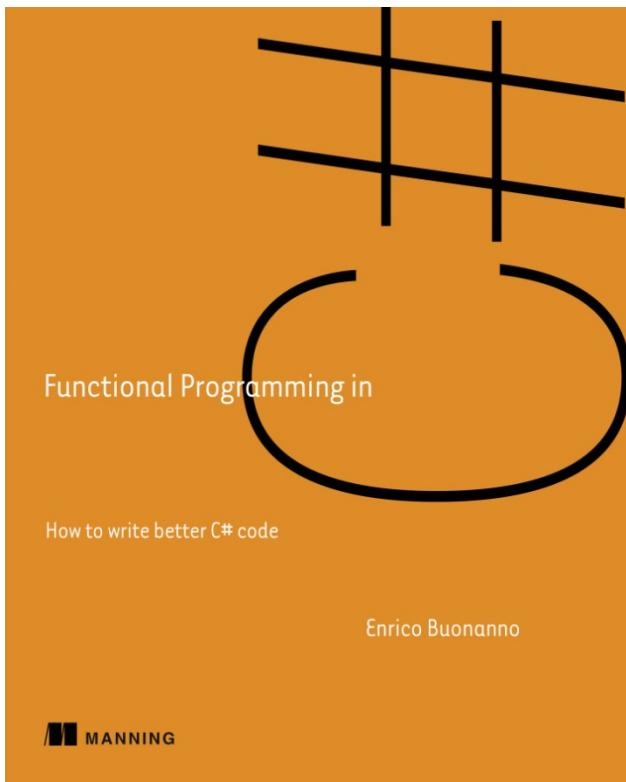
SelectMany (9)

```
var godziny =
    plan.Descendants("dzientyg")
        .Where(dzien => dzien.Value == "PoniedziaŁek")
        .Select(dzien => dzien.ElementsAfterSelf().Take(16))
        .Where(jedn => jedn.Any(n => n.Value.Contains("509")))
        .SelectMany(jedn => jedn.Where(
            wezel => wezel.Name == "godz")
            ,(wezly, godzina) => godzina.Value)
    .ToList();
```

SelectMany (10)

```
var godziny = (from dzien in root.Descendants("dzientyg")
               where dzien.Value == "PoniedziaŁek"
               select dzien.ElementsAfterSelf().Take(16)
               into pon
               where pon.Any(d => d.Value.Contains("509"))
               select pon
               into ponI509
               from wezel in ponI509
               where wezel.Name == "godz"
               select wezel.Value
)
.ToList();
```

PODSTAWOWE WZORCE PROGRAMOWANIA FUNKCYJNEGO W C#



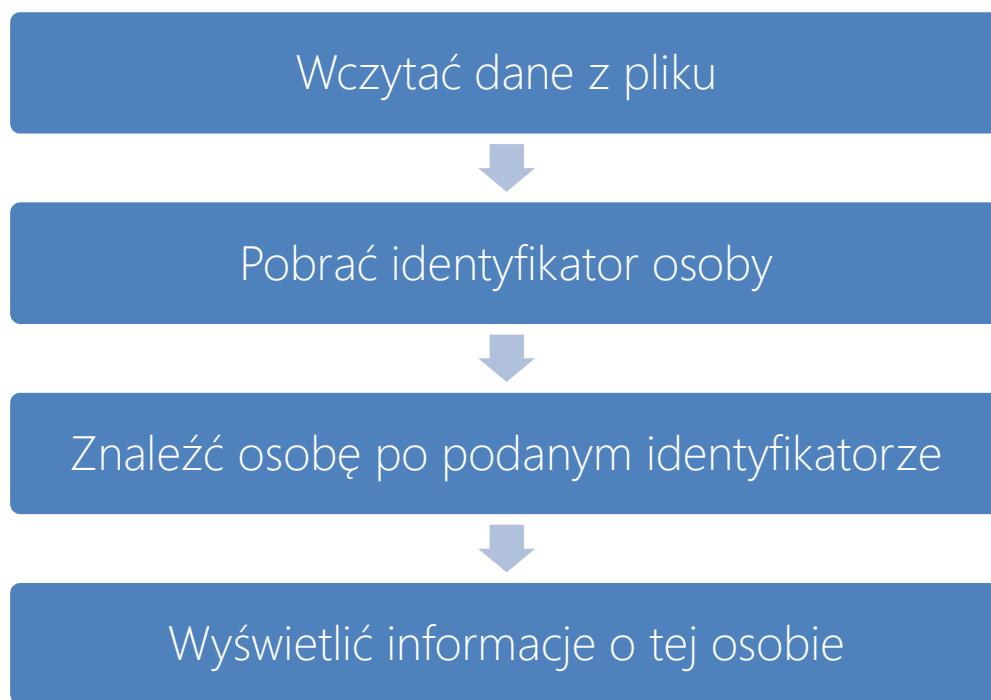
Functional Programming in C#

How to write better C# code

Enrico Buonanno

Manning Publications 2017

Przykładowa aplikacja



Przykładowa aplikacja

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyświetlDane(osoba);
    Console.ReadKey();
}
```

Przykładowa aplikacja

```
private static List<Osoba> WczytajDane(string nazwaPliku)
{
    var linie = System.IO.File.ReadAllText(nazwaPliku);
    return JsonConvert.DeserializeObject<List<Osoba>>(linie);
}

private static void WyświetlDane(Osoba osoba)
{
    Console.WriteLine($"Id: {osoba.Id}");
    Console.WriteLine($"Imię: {osoba.Imię}");
    Console.WriteLine($"Nazwisko: {osoba.Nazwisko}");
}
```

Przykładowa aplikacja

```
private static Osoba ZnajdzOsobePoId(List<Osoba> dane, int id)
{
    foreach(var osoba in dane)
    {
        if (osoba.Id == id)
            return osoba;
    }
    return null;
}
```

Co w tej aplikacji może pójść źle?



```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyświetlDane(osoba);
    Console.ReadKey();
}
```

Co w tej aplikacji może pójść źle?

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = dane[id];
    WyswietlDane(osoba);
    Console.ReadLine();
}
```

! Może rzucić 9 różnych typów wyjątków

```
private static List<Osoba> WczytajDane(string nazwaPliku)
{
    var linie = System.IO.File.ReadAllText(nazwaPliku);
    return JsonConvert.DeserializeObject<List<Osoba>>(linie);
}
```

Może rzucić wyjątek JsonReaderException

Co w tej aplikacji może pójść źle?

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyswietlDane(osoba);
    Console.ReadKey();
}
```

Mogą być rzucone wyjątki, ale są mało prawdopodobne

Co w tej aplikacji może pójść źle?

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyswietlDane(osoba);
    Console.ReadKey();
}
```

String może być w
złym formacie

Co w tej aplikacji może pójść źle?

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyswietlDane(osoba);
    Console.ReadKey();
}
```

Ten fragment jest
bezpieczny

Co w tej aplikacji może pójść źle?

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyswietlDane(osoba);
    Console.ReadKey();
}
```

Jeżeli osoba będzie null to będzie wyjątek

```
private static void WyswietlDane(Osoba osoba)
{
    Console.WriteLine($"Id: {osoba.Id}");
    Console.WriteLine($"Imie: {osoba.Imie}");
    Console.WriteLine($"Nazwisko: {osoba.Nazwisko}");
}
```

Naprawa aplikacji

```
static void Main(string[] args)
{
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
        }
```

Naprawa aplikacji

```
        catch (Exception e) {
            Console.Clear();
            PokazBlad("To musi być liczba");
            ok = false;
        }
    }
    while (!ok);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    if (osoba != null)
        WyswietlDane(osoba);
    else
        PokazBlad("Nie ma osoby o takim ID");
}
catch(Exception exception)
{
    PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
}
}
```

Naprawa aplikacji

```
private static void PokazBlad(string wiadomosc)
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine(wiadomosc);
    Console.ResetColor();
}
```

```
static void Main(string[] args) {
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
            catch (Exception e) {
                Console.Clear();
                PokazBlad("To musi być liczba");
                ok = false;
            }
        }
        while (!ok);
        Osoba osoba = ZnajdzOsobePoId(dane, id);
        if (osoba != null)
            WyswietlDane(osoba);
        else
            PokazBlad("Nie ma osoby o takim ID");
    }
    catch(Exception exception) {
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    }
}
```

```
static void Main(string[] args) {
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
            catch (Exception e) {
                Console.Clear();
                PokazBlad("To musi być liczba");
                ok = false;
            }
        }
        while (!ok);
        Osoba osoba = ZnajdzOsobePoId(dane, id);
        if (osoba != null)
            WyświetlDane(osoba);
        else
            PokazBlad("Nie ma osoby o takim ID");
    }
    catch(Exception exception) {
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    }
}
```

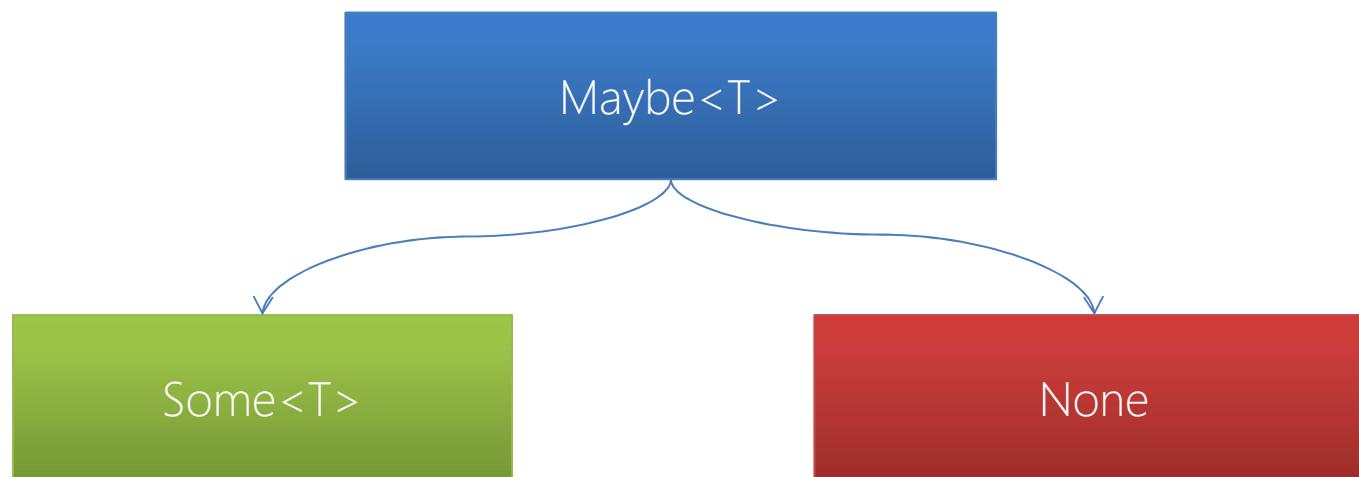


Czy można to napisać lepiej?

OBSŁUGA WARTOŚCI NULL

Typ Maybe/Option

Typ Maybe pozwala na określanie czy dana wartość istnieje czy nie



Typ Maybe

```
namespace Maybe {

    public class None {
        private None() { }
        public static readonly None Value = new None();
    }

    public class Some<T> {
        public readonly T Value;

        public Some(T value) {
            if (value == null) throw new ArgumentNullException();
            Value = value;
        }
    }
}
```

Typ Maybe

```
public static class Functional
{
    public static Maybe.None None => Maybe.None.Value;
    public static Maybe.Some<T> Some<T>(T value)
        => new Maybe.Some<T>(value);
}
```

Typ Maybe

```
namespace ConsoleApp5
{
    using static ConsoleApp5.Functional;

    class Program {
        static void Main(string[] args)
        {
            var wiek = Some(12);
            var imie = None;
        }

    }
}
```

Typ Maybe

```
public class Maybe<T> {
    readonly bool _hasValue;
    readonly T _value;

    private Maybe(T value) {
        _hasValue = true;
        _value = value;
    }

    private Maybe() { _hasValue = false; }
```

Typ Maybe

```
public static implicit operator Maybe<T>(Maybe.None v)
=> new Maybe<T>();
```

```
public static implicit operator Maybe<T>(Maybe.Some<T> v)
=> new Maybe<T>(v.Value);
```

```
public static implicit operator Maybe<T>(T v)
=> v == null ? (Maybe<T>)None : Some(v);
```

Typ Maybe

```
public TOutput When<TOutput>(
    Func<TOutput> whenNone,
    Func<T, TOutput> whenSome)
=> _hasValue ? whenSome(_value) : whenNone();

public T Reduce(Func<T> whenNone)
    => _hasValue ? _value : whenNone();
}
```

```
static void Main(string[] args) {
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
            catch (Exception e) {
                Console.Clear();
                PokazBlad("To musi być liczba");
                ok = false;
            }
        }
        while (!ok);
        Osoba osoba = ZnajdzOsobePoId(dane, id);
        if (osoba != null)
            WyswietlDane(osoba);
        else
            PokazBlad("Nie ma osoby o takim ID");
    }
    catch(Exception exception) {
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    }
}
```

```
private static Maybe<Osoba> ZnajdzOsobePoId(
    List<Osoba> dane, int id)
{
    foreach (var osoba in dane) {
        if (osoba.Id == id)
            return osoba;
    }
    return None;
}
```

```
Osoba osoba = ZnajdzOsobePoId(dane, id);
if (osoba != null)
    WyswietlDane(osoba);
else
    PokazBlad("Nie ma osoby o takim ID");
```



```
var osoba = ZnajdzOsobePoId(dane, id );
osoba.When(
    some: ...,
    none: ...
);
```

```
internal class Osoba
{
    public int Id { get; set; }
    public string Imie { get; set; }
    public string Nazwisko { get; set; }

    public override string ToString() {
        return new StringBuilder()
            .AppendLine($"Id: {Id}")
            .AppendLine($"Imie: {Imie}")
            .AppendLine($"Nazwisko: {Nazwisko}").ToString();
    }
}
```

```
Osoba osoba = ZnajdzOsobePoId(dane, id);
if (osoba != null)
    WyswietlDane(osoba);
else
    PokazBlad("Nie ma osoby o takim ID");
```



```
var osoba = ZnajdzOsobePoId(dane, id);
Console.WriteLine(
    osoba.When(
        some: (o) => o.ToString(),
        none: ()   => "Nie ma osoby o takim ID"
    )
);
```

```
static void Main(string[] args) {
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
            catch (Exception e) {
                Console.Clear();
                PokazBlad("To musi być liczba");
                ok = false;
            }
        }
        while (!ok);
        var osoba = ZnajdzOsobePoId(dane, id);
        Console.WriteLine(
            osoba.When( some: (o) => o.ToString(), none: ()  => "Nie ma osoby o takim ID" )
        );
    }
    catch(Exception exception) {
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    }
}
```

```
public static class Extensions {
    public static Maybe<int> ParseToInt(this string str) {
        try {
            return int.Parse(str);
        }
        catch(Exception) {
            return None;
        }
    }
}
```

```
Maybe<int> id;
do {
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    id = str.ParseToInt();
}

while (id == (Maybe<int>)None);

var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0));
```

```
static void Main(string[] args) {
    try {
        var dane = WczytajDane("dane.json");
        Maybe<int> id;

        do {
            Console.WriteLine("Podaj ID osoby do wyświetlenia");
            string str = Console.ReadLine();
            id = str.ParseToInt();
        }
        while (id == (Maybe<int>)None);

        var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0) );
        Console.WriteLine(
            osoba.When( some: (o) => o.ToString(), none: ()  => "Nie ma osoby o takim ID" )
        );
    }
    catch(Exception exception) {
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    }
}
```

```
private static Maybe<List<Osoba>> WczytajDane(string nazwaPliku)
{
    try {
        var linie = System.IO.File.ReadAllText(nazwaPliku);
        return JsonConvert.DeserializeObject<List<Osoba>>(linie);
    }
    catch(Exception e) {
        return None;
    }
}
```

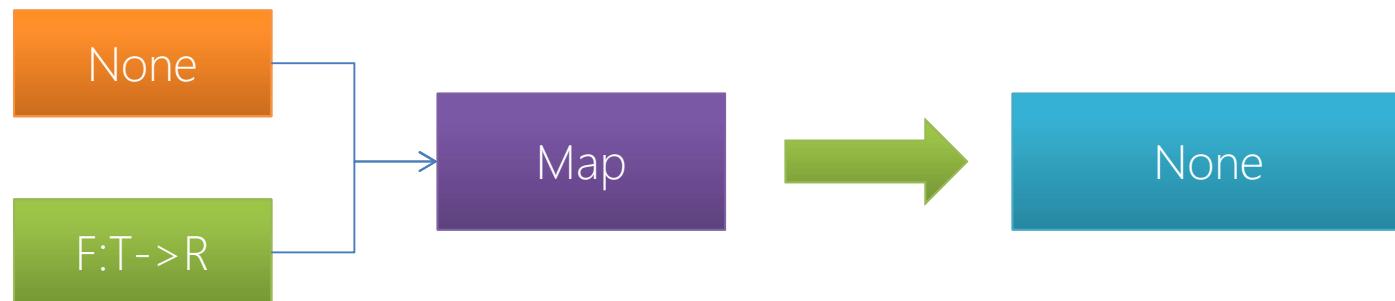
```
static void Main(string[] args) {
    var wynik = WczytajDane("dane.json")
        .When( some: (dane) =>
    {
        Maybe<int> id;
        do {
            Console.WriteLine("Podaj ID osoby do wyświetlenia");
            string str = Console.ReadLine();
            id = str.ParseToInt();
        }
        while (id == (Maybe<int>)None);
        var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0));
        return osoba.When(
            some: (o) => o.ToString(),
            none: () => "Nie ma osoby o takim ID"
        );
    },
    none: () => "Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    Console.WriteLine(wynik);
}
```

MAPOWANIE, FUNKTORY I MONADY

Mapowanie

$$\text{Map}: (M < T >, (T \rightarrow R)) \rightarrow M < R >$$
$$\text{Map}: (\text{Maybe} < T >, (T \rightarrow R)) \rightarrow \text{Maybe} < R >$$

Mapowanie



Mapowanie

```
public static class Functional
{
    public static Maybe<R> Map<T, R>(
        this Maybe<T> maybe, Func<T, R> map)
        => maybe.When(
            () => (Maybe<R>)None,
            (v) => Some(map(v))
        )
    );
}
```

```
static void Main(string[] args) {
    var wynik =
        WczytajDane("dane.json")
            .Map((dane) => {
                Maybe<int> id;
                do
                    id = WczytajId();
                    while (id == (Maybe<int>)None);
                    var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0));
                    return osoba.When(
                        none: ()=>"Nie ma osoby o takim ID",
                        some: o => o.ToString());
            });
    Console.WriteLine(wynik.Reduce(()=>""));
    Console.ReadKey();
}
```

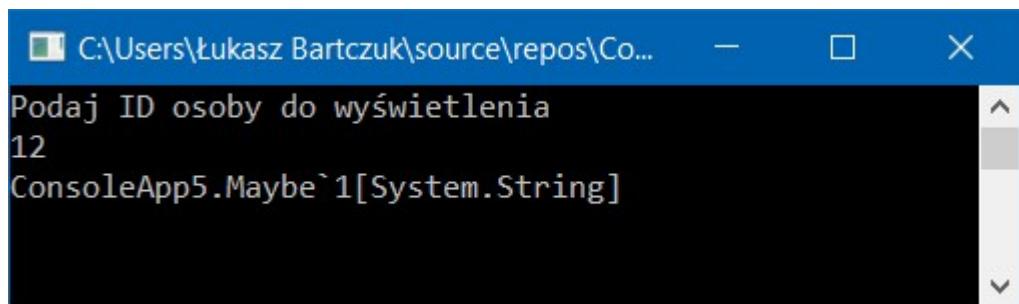
Funktory

$$\text{Map}: (M < T >, (T \rightarrow R)) \rightarrow M < R >$$

W programowaniu funkcyjnym dowolny typ wyposażony w funkcję Map nazywamy funktem

Mapowania

```
var wynik =
    WczytajDane("dane.json")
        .Map((dane) => {
            Maybe<int> id;
            do
                id = WczytajId();
            while (id == (Maybe<int>)None);
            var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0));
            return osoba.Map(o => o.ToString());
        });
    Console.WriteLine(wynik.Reduce(()=>"""));
    Console.ReadKey();
```



Mapowania

```
var wynik =
    WczytajDane("dane.json")
        .Map((dane) => {
            Maybe<int> id;
            do
                id = WczytajId();
            while (id == (Maybe<int>)None);
            return ZnajdzOsobePoId(dane, id.Reduce(() => 0));
        })
        .Map(o=>o.ToString());
Console.WriteLine(wynik.Reduce(()=>""));
Console.ReadKey();
```

Bindowanie

Bind: $(M < T >, (T \rightarrow M < R >)) \rightarrow M < R >$

Bind: $(\text{Maybe} < T >, (T \rightarrow \text{Maybe} < R >)) \rightarrow \text{Maybe} < R >$

Monady

Return: $T \rightarrow M < T >$

Return: $(T) \rightarrow \text{Maybe} < T >$

W programowaniu funkcyjnym dowolny typ wyposażony w funkcje Bind i Return nazywamy monadą

Bindowanie

```
public static class Functional
{
    public static Maybe<R> Map<T, R>(
        this Maybe<T> maybe,
        Func<T, R> map)
        => maybe.When(() => (Maybe<R>)None,
                      (v) => Some(map(v)));

    public static Maybe<R> Bind<T, R>(
        this Maybe<T> maybe,
        Func<T, Maybe<R>> map)
        => maybe.When(() => (Maybe<R>)None,
                      (v) => map(v));
}
```

Bindowanie

```
public static class Functional
{
    public static Maybe<R> Map<T, R>(
        this Maybe<T> maybe,
        Func<T, R> map)
        => maybe.When(() => (Maybe<R>)None,
                      (v) => Some(map(v)));

    public static Maybe<R> Bind<T, R>(
        this Maybe<T> maybe,
        Func<T, Maybe<R>> map)
        => maybe.When(() => (Maybe<R>)None,
                      (v) => map(v));
}
```

Bindowanie

```
var wynik =
    WczytajDane("dane.json")

    .Bind((dane) => {

        Maybe<int> id;
        do
            id = WczytajId();
        while (id == (Maybe<int>)None);
        return ZnajdzOsobePoId(dane, id.Reduce(() => 0));
    })
    .Map(o=>o.ToString());
Console.WriteLine(wynik.Reduce(()=>""));
Console.ReadKey();
```

Bindowanie

```
var wynik =
    WczytajDane("dane.json")
        .Bind((dane) => {
            Maybe<int> id;
            do
                id = WczytajId();
            while (id == (Maybe<int>)None);
            return id.Map((i) => (dane:dane, id:i));
        })
        .Bind((t)=>ZnajdzOsobePoId(t.dane,t.id ))
        .Map(o=>o.ToString());
Console.WriteLine(wynik.Reduce(()=>""));
Console.ReadKey();
```

Bindowanie

```
private static Maybe<(List<Osoba> dane, int id)>
    WczytajID(List<Osoba> dane)
{
    Maybe<int> id;
    do
        id = WczytajId();
    while (id == (Maybe<int>)None);
    return id.Map((i) => (dane: dane, id: i));
}
```

Bindowanie

```
static void Main(string[] args)
{
    var wynik =
        WczytajDane("dane.json")
            .Bind((dane)=>WczytajID(dane))
            .Bind((t) => ZnajdzOsobePoId(t.dane, t.id))
            .Map(o => o.ToString());
    Console.WriteLine(wynik.Reduce(() => ""));
    Console.ReadKey();
}
```

MONADA EITHER

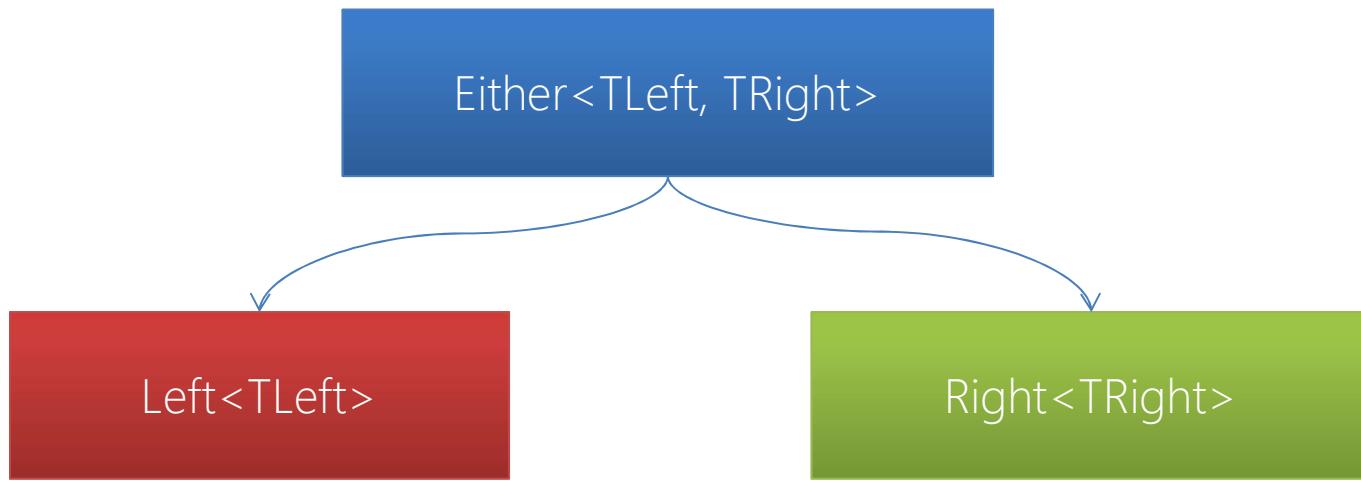
Either

Monada Either jest funkcyjną metodą pozwalającą na rozwiązywanie problemu obsługi błędów

Realizacja Railway Oriented Programming



Monada Either



Najczęściej przenosi
informacje o błędzie

Najczęściej przenosi
informacje poprawne

Monada Either

```
namespace Either {  
  
    public class Left<TLeft>{  
        public readonly TLeft Value;  
  
        public Left(TLeft value) {  
            if (value == null)  
                throw new ArgumentNullException();  
            Value = value;  
        }  
    }  
  
    public class Right<TRight> { ... }  
}
```



Implementacja taka
jak Left

Monada Either

```
public class Either<TLeft, TRight> {
    TLeft _left;
    TRight _right;
    public readonly bool IsRight;
    public readonly bool IsLeft;

    private Either(TLeft value) {
        _left = value;
        IsLeft = true; IsRight = false;
    }

    private Either(TRight value)
        _right = value;
        IsLeft = false; IsRight = true;
    }
```

Monada Either

```
public TResult Match<TResult>(
    Func<TLeft, TResult> leftFunction,
    Func<TRight, TResult> rightFunction)
    => IsLeft ? leftFunction(this._left) :
        rightFunction(this._right);

public void Do(Action<TLeft> leftAction,
               Action<TRight> rightAction) {
    if (IsLeft)
        leftAction(this._left);
    else
        rightAction(this._right);
}

public void WhenLeft(Action<TLeft> a) { if(IsLeft) a(_left); }
```

Monada Either

```
public static implicit operator Either<TL, TR>(Either.Left<TL> v)
    => new Either<TL, TR>(v.Value);

public static implicit operator Either<TL, TR>(Either.Right<TR> v)
    => new Either<TL, TR>(v.Value);

public static implicit operator Either<TL, TR>(TL v)
    => new Either<TL, TR>(v);

public static implicit operator Either<TL, TR>(TR v)
    => new Either<TL, TR>(v);
}
```

Monada Either

```
public static class Functional {

    public static Either.Left<L> Left<L>(L v)
        => new Either.Left<L>(v);
    public static Either.Right<R> Right<R>(R v)
        => new Either.Right<R>(v);

    public static Either<TNL, TNR> Map<TL, TR, TNL, TNR>
        (this Either<TL, TR> either,
         Func<TL, TNL> mapLeft,
         Func<TR, TNR> mapRight)
        => either.Match<Either<TNL, TNR>>
            (l => Left(mapLeft(l)), r => Right(mapRight(r)));
}
```

Monada Either

```
public static Either<TL, TNR>
Map<TL, TR, TNR>
(this Either<TL, TR> either,
Func<TR, TNR> map)
=> either.Match<Either<TL, TNR>>(l => Left(l), r => Right(map(r)));


public static Either<TL, TNR>
Bind<TL, TR, TNR>
(this Either<TL, TR> either,
Func<TR, Either<TL, TNR>> map)
=> either.Match(l => Left(l), r => map(r));
}
```

Przykładowy program

```
static void Main(string[] args)
{
    var wynik =
        WczytajDane("dane.json")
            .Bind((dane)=>WczytajID(dane))
            .Bind((t) => ZnajdzOsobePoId(t.dane, t.id))
            .Map(o => o.ToString());
    Console.WriteLine(wynik.Reduce(() => ""));
    Console.ReadKey();
}
```

Monada Either

```
private static Either<string, List<Osoba>>
    WczytajDane(string nazwaPliku)
{
    try {
        var linie = File.ReadAllText(nazwaPliku);
        return JsonConvert.DeserializeObject<List<Osoba>>(linie);
    }
    catch (Exception e) {
        return e.Message;
    }
}
```

Monada Either

```
private static Either<string, (List<Osoba> dane, int id)>
    WczytajID(List<Osoba> dane) {
    Either<string, int> id;
    do {
        id = WczytajId();
        id.WhenLeft(PokazBlad);
    }
    while ( id.IsLeft );
    return id.Map((s)=>s, (i) => (dane, i));
}

private static Either<string, int> WczytajId() {
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    return str.ParseToInt();
}
```

Monada Either

```
public static class Extensions {
    public static Either<string,int> ParseToInt(this string str)
    {
        try {
            return int.Parse(str);
        }
        catch (Exception e) {
            return e.Message;
        }
    }
}
```

Monada Either

```
class Program {  
  
    public static Either<string, Maybe<Osoba>>  
        ZnajdzOsobe((List<Osoba> dane, int id) t)  
        => ZnajdzOsobePoId(t.dane, t.id);  
  
    public static Either<string, string> Redukuj(Maybe<Osoba> o)  
        => Right(o.Map(os => os.ToString())  
            .Reduce(() => "Brak osoby o takim id"));
```

Monada Either

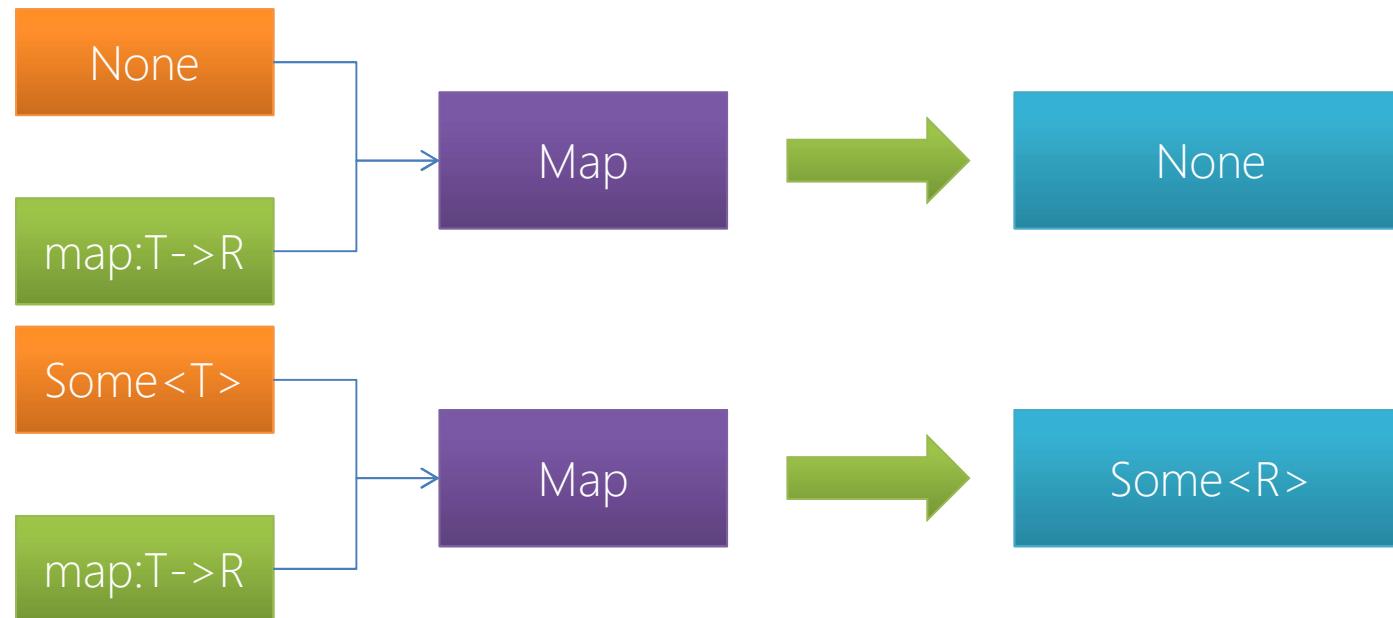
```
static void Main(string[] args) {
    WczytajDane("dane.json")
        .Bind((dane) => WczytajID(dane))
        .Bind(ZnajdzOsobe)
        .Bind(Redukuj)
        .Do(
            right: Console.WriteLine,
            left: PokazBlad
        );
    Console.ReadKey();
}
```

SelectMany (11)

```
public static class Functional {  
  
    public static Maybe<R> Map<T, R>(  
        this Maybe<T> maybe, Func<T, R> map)  
        => maybe.When(() => (Maybe<R>)None, (v) => Some(map(v)));  
  
    public static Maybe<R> Bind<T, R>(  
        this Maybe<T> maybe, Func<T, Maybe<R>> map)  
        => maybe.When(() => (Maybe<R>)None, (v) => map(v));  
}
```

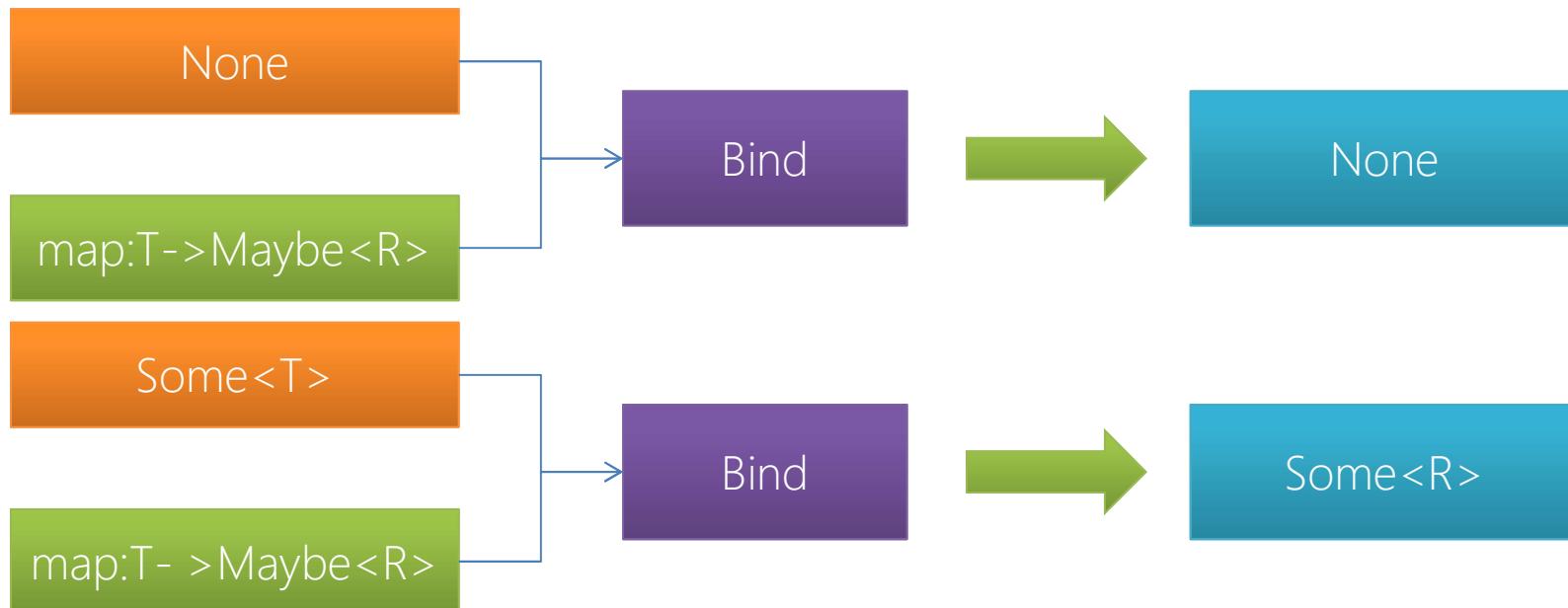
Map

```
public static Maybe<R> Map<T, R>(  
    this Maybe<T> maybe, Func<T, R> map)  
    => maybe.When(() => (Maybe<R>)None, (v) => Some(map(v)));
```



Bind

```
public static Maybe<R> Bind<T, R>(  
    this Maybe<T> maybe, Func<T, Maybe<R>> map)  
=> maybe.When(() => (Maybe<R>)None, (v) => map(v));
```



SelectMany (12)

```
static Maybe<R> Bind<T, R>(
    this Maybe<T> maybe,
    Func<T, Maybe<R>> map)
```

```
static IEnumerable<R> SelectMany<A, R>(
    this IEnumerable<A> sequence,
    Func<A, IEnumerable<R>> function)
```

SelectMany (13)

```
var wynik =
    WczytajDane("dane.json")
        .Bind((dane) => WczytajID(dane))
        .Bind((t) => ZnajdzOsobePoId(t.dane, t.id))
        .Select(o => o.ToString());
Console.WriteLine(wynik.Reduce(() => ""));
```

SelectMany (14)

```
public static class Functional {  
  
    public static Maybe<S> SelectMany<T, R, S>(  
        this Maybe<T> maybe,  
        Func<T, Maybe<R>> map,  
        Func<T, R, S> resultSelector)  
        => maybe.Bind(  
            outer => map(outer).Bind(  
                inner => (Maybe<S>)Some(  
                    resultSelector(outer, inner))));  
  
}
```

SelectMany (15)

```
var osoba = WczytajDane("dane.json")
    .SelectMany(dane => WczytajID(),
                (dane, id) => (dane: dane, id: id))
    .SelectMany(d => ZnajdzOsobePoId(d.dane, d.id),
                (d, o) => o.ToString());  
  
Console.WriteLine(osoba.Reduce(() => ""));
```

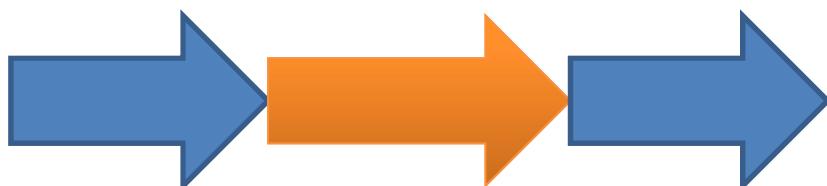
SelectMany (16)

```
var osoba =  
    from dane in WczytajDane("dane.json")  
    from id in WczytajID()  
    from osoba in ZnajdzOsobePoId(dane, id)  
    select osoba.ToString();  
  
Console.WriteLine(osoba.Reduce(() => ""));
```

PROGRAMOWANIE ASYNCHRONICZNE

Programowanie synchroniczne

Programowanie synchroniczne polega na wykonywaniu poleceń w kolejności jedno po drugim



Programowanie synchroniczne

Przykład 1:

```
private void Komunikat(string wiadomosc) {
    MessageBox.Show(${wiadomosc});
}

private void button1_Click(object sender, EventArgs e)
{
    Stopwatch sw = new Stopwatch();
    Komunikat("Button1_click: Początek");
    sw.Start();
    var wynik = PobierzDaneOPogodzie("synchronicznie");
    sw.Stop();
    Komunikat($"Wynik: {wynik} po: {sw.ElapsedMilliseconds}");
    Komunikat("Button1_click: Koniec");
}
```

Programowanie synchroniczne

Przykład 1:

```
private string PobierzDaneOPogodzie(object argument)
{
    Thread.Sleep(5000);
    return "Słonecznie";
}
```

Programowanie synchroniczne

Przykład 2:

```
private string PobierzDaneOWalutach(object argument)
{
    Thread.Sleep(10000);
    return "Dolar bez zmian";
}

private string PobierzWiadomosci(object argument)
{
    Thread.Sleep(1000);
    return "Wiadomości są dobre";
}
```

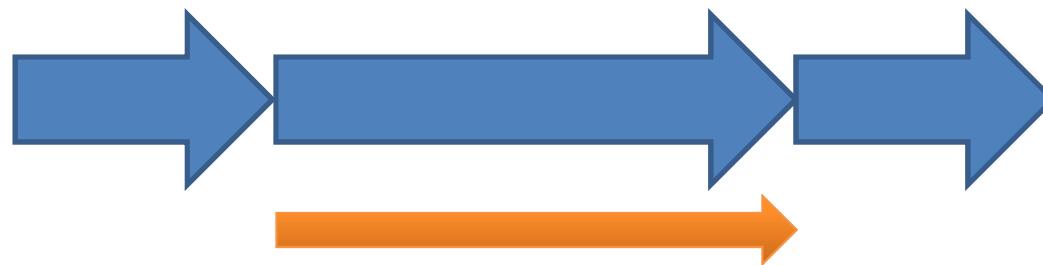
Programowanie synchroniczne

Przykład 2:

```
private void button2_Click(object sender, EventArgs e)
{
    Stopwatch sw = new Stopwatch();
    Komunikat("Button2_click: Początek");
    sw.Start();
    var wynik1 = PobierzDaneOPogodzie("synchronicznie");
    var wynik2 = PobierzDaneOWalutach("synchronicznie");
    var wynik3 = PobierzWiadomosci("synchronicznie");
    sw.Stop();
    Komunikat($"Wyniki po: {sw.ElapsedMilliseconds}");
    Komunikat("Button2_click: Koniec");
}
```

Programowanie asynchroniczne

Zadania są wykonywane niezależnie (całość nie jest blokowana), przy czym poszczególne zadania mogą być (choć nie muszą) wykonywane w różnych wątkach.



Programowanie asynchroniczne

W celu ułatwienia tworzenia aplikacji asynchronicznych w języku C# dostępne są dwie klasy

Task

Task<TResult>

Klasa Task

Klasa Task reprezentuje operację asynchroniczną,
która nie zwraca wartości

Metoda	Opis
Start	Rozpoczyna wykonywanie zadania
Wait	Czeka aż wątek zakończy wykonywanie swojego działania
WaitAll(Task[])	Czeka na zakończenie wszystkich działań
WaitAny(Task[])	Czeka na zakończenie dowolnego działania
Delay(int32)	Tworzy zadanie, które zakończy się po określonym czasie

Klasa Task

Klasa Task reprezentuje operację asynchroniczną,
która nie zwraca wartości

Metoda	Opis
ContinueWith(Action<Task>)	Tworzy kontynuację, która będzie uruchomiona w momencie zakończenia zadania
WhenAll(Task[])	Tworzy zadanie, które zakończy się jeżeli wszystkie zadania w tablicy się zakończą
WhenAny(Task[])	Tworzy zadanie, które zakończy się jeżeli jakiekolwiek zadanie z tablicy się zakończy

Klasa Task<TResult>

Klasa Task<TResult> reprezentuje operację asynchroniczną, która zwraca wartość typu TResult

Posiada ona podobne metody, co klasa Task, a dodatkowo zawiera właściwość:

TResult Result

która pozwala odczytać rezultat operacji asynchronicznej

Programowanie asynchroniczne

Przykład 3:

```
private void Komunikat(string wiadomosc)
{
    string taskId = Task.CurrentId.HasValue ?
        Task.CurrentId.ToString() : "UI";

    int threadId = Thread.CurrentThread.ManagedThreadId;

    MessageBox.Show($"{wiadomosc} ({taskId}) {threadId}");
}
```

Programowanie asynchroniczne

Przykład 3:

```
private void button1_Click(object sender, EventArgs e) {
    Komunikat("Button1_click: Początek");

    var zadanie = new Task<string>(PobierzDaneOPogodzie, "zadanie");
    zadanie.Start();

    Komunikat("Akcja została uruchomiona");

    if (zadanie.Status != TaskStatus.Running &&
        zadanie.Status != TaskStatus.RanToCompletion)
        Komunikat("Zadanie nie zostało uruchomione");
    else
        Komunikat($"Wynik: " + zadanie.Result);
    Komunikat("Button1_click: koniec");
}
```

Programowanie asynchroniczne

Przykład 4:

```
private void button2_Click(object sender, EventArgs e) {
    Komunikat("Button2_click: Początek");
    Task<string> zadanie =
        new Task<string>(PobierzDaneOPogodzie, "zadanie");

    zadanie.ContinueWith(poprzednieZadanie => {
        Komunikat($"Wynik: " + poprzednieZadanie.Result);
    });

    zadanie.Start();
    Komunikat("Button3_click: koniec");
}
```

Programowanie synchroniczne

Przykład 5:

```
private void Kontynuacja(Task<string> zadanie) {
    Komunikat($"Wynik: " + zadanie.Result);
}

private void button3_Click(object sender, EventArgs e) {
    Komunikat("Button3_Click: Początek");
    Task<string> zadanie =
        new Task<string>(PobierzDaneOPogodzie, "zadanie");

    zadanie.ContinueWith(Kontynuacja);
    zadanie.Start();
}
```

Programowanie asynchroniczne

Przykład 6:

```
private Task<string> OperacjaAsync() {
    Task<string> zadanie =
        new Task<string>(PobierzDaneOPogodzie, "zadanie");

    zadanie.Start();
    return zadanie;
}

private void button4_Click(object sender, EventArgs e) {
    Komunikat("Button4_click: Początek");
    var zadanie = OperacjaAsync();

    zadanie.ContinueWith(Kontynuacja);
}
```

async/await

async / await są operatorami języka C#, które ułatwiają tworzenie operacji asynchronousznych.

Dzięki nim możliwe jest tworzenie programów asynchronousznych w sposób jak najbardziej zbliżony do programów synchronicznych

async/await

Przykład 7:

```
private async void button5_Click(object sender, EventArgs e)
{
    Komunikat("Button5_Click: Początek");
    string wynik = await OperacjaAsync();
    Komunikat($"Wynik: " + wynik);
    Komunikat("Button5_Click: koniec");
}
```

programowanie współbieżne

W programowaniu współbieżnym kilka zadań jest wykonywanych równocześnie.



programowanie współbieżne

```
private Task<string> PobierzDaneOPogodzieAsync() {
    var zadanie = new Task<string>(PobierzDaneOPogodzie, "Pogoda");
    zadanie.Start();
    return zadanie;
}
private Task<string> PobierzDaneOWalutachAsync() {
    var zadanie = new Task<string>(PobierzDaneOWalutach, "Waluty");
    zadanie.Start();
    return zadanie;
}
private Task<string> PobierzDaneWiadomosciAsync() {
    var zadanie = new Task<string>(PobierzWiadomosci, "Wiadomości");
    zadanie.Start();
    return zadanie;
}
```

programowanie współbieżne

```
private void button6_Click(object sender, EventArgs e) {  
  
    Komunikat("button6_click: Poczatek");  
    Stopwatch sw = new Stopwatch();  
    sw.Start();  
    var zadanie1 = PobierzDaneOWalutachAsync();  
    var zadanie2 = PobierzDaneOPogodzieAsync();  
    var zadanie3 = PobierzDaneWiadomosciAsync();  
  
    Task.WaitAll(zadanie1, zadanie2, zadanie3);  
    sw.Stop();  
    Komunikat("button6_click: Koniec " + sw.ElapsedMilliseconds);  
}
```

programowanie współbieżne

```
private Task Zadanie()
{
    var zadanie = new Task(() =>
    {
        var zadanie1 = PobierzDaneOWalutachAsync();
        var zadanie2 = PobierzDaneOPogodzieAsync();
        var zadanie3 = PobierzDaneWiadomosciAsync();

        Task.WaitAll(zadanie1, zadanie2, zadanie3);
    });

    zadanie.Start();
    return zadanie;
}
```

programowanie współbieżne

```
private async void button7_Click(object sender, EventArgs e)
{
    Komunikat("button7_click: Początek");
    Stopwatch sw = new Stopwatch();
    sw.Start();
    await Zadanie();
    sw.Stop();
    Komunikat("button7_click: Koniec "+sw.ElapsedMilliseconds);
}
```

programowanie współbieżne ze stanem

```
private void Wykonaj(int od, int @do) {
    for (int i = od; i <= @do; i++) wynik++;
    System.Threading.Thread.Sleep(1000);
}
private void button8_Click(object sender, EventArgs e) {
    wynik = 0;
    Stopwatch sw = new Stopwatch();
    sw.Start();
    Wykonaj(    1,  250000);
    Wykonaj(250001,  500000);
    Wykonaj(500001,  750000);
    Wykonaj(750001, 1000000);
    sw.Stop();
    WynikLabel.Text = wynik.ToString();
    CzasLabel.Text = sw.ElapsedMilliseconds.ToString(); }
```

programowanie współbieżne ze stanem

```
private int wynik;
private void Wykonaj(int od, int @do) {
    for (int i = od; i <= @do; i++) wynik++;
    System.Threading.Thread.Sleep(1000);
}
private void button8_Click(object sender, EventArgs e) {
    wynik = 0;
    Stopwatch sw = new Stopwatch();
    sw.Start();
    Wykonaj(    1,  250000);
    Wykonaj(250001,  500000);
    Wykonaj(500001,  750000);
    Wykonaj(750001, 1000000);
    sw.Stop();
    WynikLabel.Text = wynik.ToString();
    CzasLabel.Text = sw.ElapsedMilliseconds.ToString(); }
```

programowanie współbieżne ze stanem

```
private void button9_Click(object sender, EventArgs e) {
    wynik = 0;
    Stopwatch sw = new Stopwatch();
    Task[] t = new[] { new Task(()=>Wykonaj(1, 250000)),
                      new Task(()=>Wykonaj(25001, 500000)),
                      new Task(()=>Wykonaj(50001, 750000)),
                      new Task(()=>Wykonaj(75001, 1000000))};
    sw.Start();
    for (int i = 0; i < t.Length; i++) t[i].Start();
    Task.WaitAll(t);
    sw.Stop();
    label1.Text = wynik.ToString();
    label2.Text = sw.ElapsedMilliseconds.ToString();
}
```

programowanie współbieżne ze stanem

```
object locker = new object();

void Wykonaj(int od, int @do)
{
    for (int i = od; i <= @do; i++)
        lock (locker)
    {
        wynik++;
    }
    System.Threading.Thread.Sleep(1000);
}
```

programowanie współbieżne bez stanu

```
int WykonajBezStanu(int od, int @do) {
    int wynik = 0;
    for (int i = od; i <= @do; i++) wynik++;
    System.Threading.Thread.Sleep(1000);
    return wynik;
}
```

programowanie współbieżne bez stanu

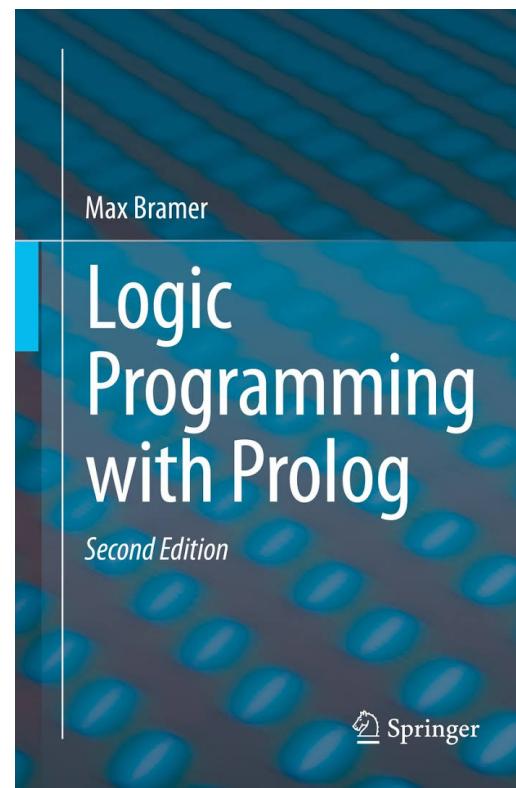
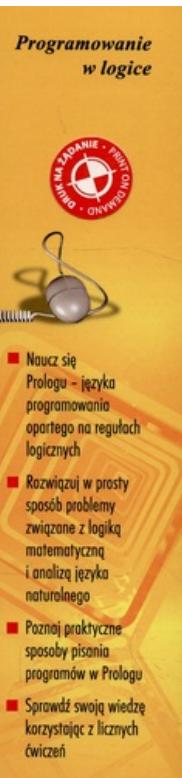
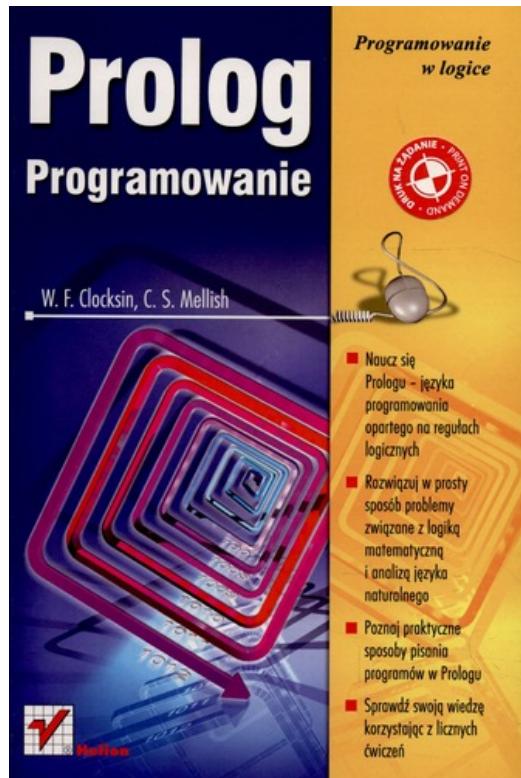
```
private void button11_Click(object sender, EventArgs e) {
    Stopwatch sw = new Stopwatch();
    var ts = new[] {new Task<int>(()=>WykonajBezStanu(      1,   250000)),
                  new Task<int>(()=>WykonajBezStanu(250001,   500000)),
                  new Task<int>(()=>WykonajBezStanu(500001,   750000)),
                  new Task<int>(()=>WykonajBezStanu(750001, 1000000))};
    sw.Start();
    for (int i = 0; i < ts.Length; i++) ts[i].Start();
    Task.WaitAll(ts);

    wynik = ts.Sum(t=>t.Result);
}

sw.Stop();
label1.Text = wynik.ToString();
label2.Text = sw.ElapsedMilliseconds.ToString(); }
```

PROGRAMOWANIE W LOGICE

Literatura



Programowanie w logice

Każdy student informatyki na PCz lubi Paradygmaty Programowania.

Tomek Nowak jest studentem informatyki na PCz.

Tomek Nowak lubi Paradygmaty Programowania!

Programowanie w logice

"Ideą programowania w logice jest wykorzystanie komputera do wyprowadzenia konkluzji z deklaratywnych opisów "

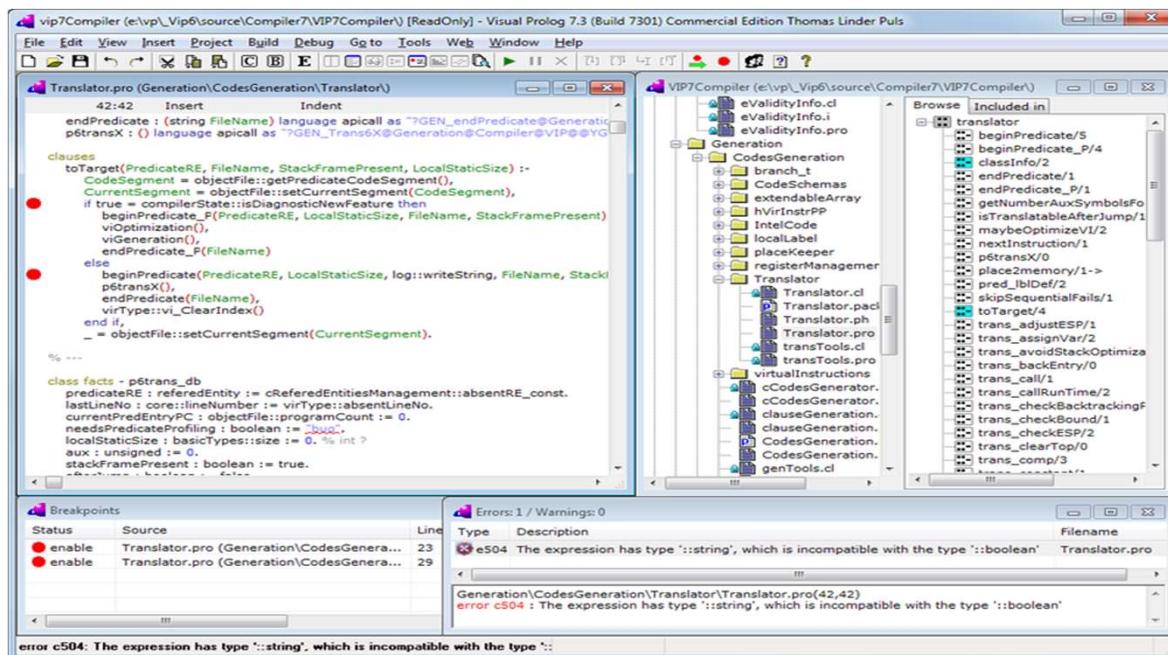
Ulf Nilsson, Jan Małuszyński "Logic, Programming and Prolog (2ED)"

PROLOG

- Język programowania stworzony w 1972 r. przez Alain Colmerauer'a i Phillipa Roussel'a, pierwotnie do rozwiązywania zagadnień związanych z przetwarzaniem języka naturalnego
- Jest realizacją paradygmatu programowania w logice
- Jest językiem deklaratywnym – pozwala na deklarowanie pewnych założeń dotyczących rozwiązywanego problemu

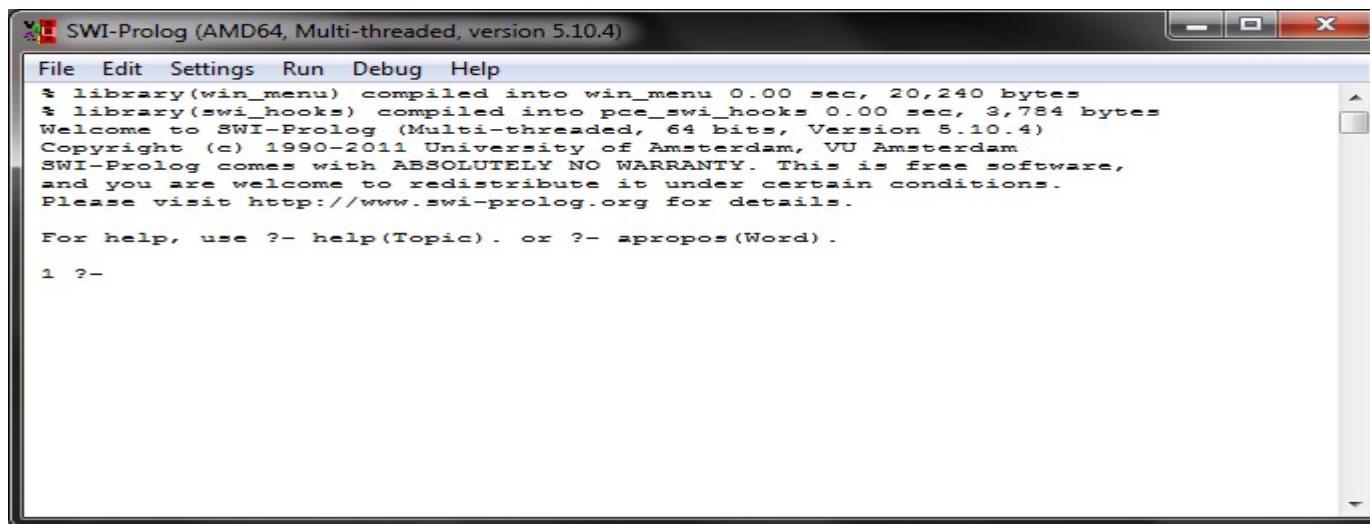
Implementacje

Visual Prolog



Oprogramowanie

SWI-Prolog



Podstawowe elementy języka PROLOG

- **Fakty** – podstawowa informacja na temat pewnego "światu" (analizowanego zagadnienia)
 - **Reguły** – zdania łączące fakty umożliwiające przeprowadzenie wnioskowania w analizowanym "świecie"
 - **Zapytania** – wydawane w celu uzyskania pewnych informacji na temat analizowanego świata
- 
- Baza
wiedzy

Fakty

Fakty – podstawowa informacja na temat pewnego "świata"
(analizowanego zagadnienia)

mezczyzna(jan).

mezczyzna(tomasz).

kobieca(alicja).

kobieca(filomena).

rodzic(filomena, tomasz).

rodzic(filomena, alicja).

rodzic(jan, tomasz).

rodzic(jan, alicja).

Reguły

Reguły – zdania łączące fakty umożliwiające przeprowadzenie wnioskowania w analizowanym "świecie"

```
ojciec(X, Y) :- rodzic(X, Y), mężczyzna(X).  
matka(X, Y) :- rodzic(X, Y), kobieta(X).  
dziadek(X, Y) :- ojciec(X,Z), ojciec(Z,Y).
```

Zmienne w PROLOG-u

Prolog jest językiem o dynamicznym systemie typów.

Zmienne w Prologu zawsze rozpoczynają się od dużej litery.

```
mezczyzna(jan).  
mezczyzna(tomasz).
```

```
ojciec(X, Y) :- rodzic(X, Y), mezczyzna(X).
```

Zapytania

Zapytania – wydawane w celu uzyskania pewnych informacji na temat analizowanego świata

mezczyszna(jan).

mezczyszna(tomasz).

kobieta(alicja).

kobieta(filomena).

rodzic(filomena, tomasz).

? - mezczyszna(jan).
true

? - mezczyszna(teofil).
false

Zapytania

mezczyzna(jan).



mezczyzna(tomek).



kobieta(ewelina).

kobieta(maria).

matka(maria, tomek).



matka(ewelina, jan).



ojciec(tomek, jan).

?- mezczyzna(tomek).



true

? - matka(ewelina, jan).



true

?- matka(ewelina, tomek).



false

Zapytania

mezczyzna(jan).



mezczyzna(tomasz).

kobieta(alicja).

kobieta(filomena).

rodzic(filomena, tomasz).



rodzic(alicja, filomena).



rodzic(jan, tomasz).



?- mezczyzna(Kto).

Kto = jan ;

Kto = tomasz.

?- rodzic(Kto, tomasz).



Kto = filomena ;

Kto = jan.

Zapytania złożone

Zapytania o pojedyncze fakty można łączyć za pomocą operatorów logicznych.

Operator logiczny	Operator Prologu
\Leftarrow (implikacja)	$:$ -
\wedge (koniunkcja)	,
\vee (alternatywa)	$;$

```
?- lubi(jan, aneta), lubi(tomek, aneta)
```

```
no
```

```
?- lubi(jan, X), lubi(tomek, X)
```

```
X=muzyka
```

Zapytania z regułami

mezczyzna(jan).

mezczyzna(tomasz).

mezczyzna(adam).

mezczyzna(tadeusz).

mezczyzna(maurycy).

mezczyzna(antoni).

rodzic(adam, tadeusz).

rodzic(adam, antoni).

rodzic(tadeusz, jan).

rodzic(jan, tomasz).

rodzic(antoni, maurycy).

ojciec(X, Y) :- rodzic(X, Y), mezczyzna(X).

matka(X, Y) :- rodzic(X, Y), kobieta(X).

dziadek(X, Y) :- ojciec(X, Z), ojciec(Z, Y).

Zapytania z regułami

```
ojciec(X, Y) :- rodzic(X, Y), mezczyzna(X). ←  
matka(X, Y) :- rodzic(X, Y), kobieta(X). ←  
dziadek(X, Y) :- ojciec(X, Z), ojciec(Z, Y). ←  
  
?- dziadek(adam, tomasz).  
      ↓  
ojciec(adam, Z), ojciec(Z, tomasz).  
      ↓  
rodzic(adam, Z), mezczyzna(adam), ojciec(Z, tomasz).
```

Zapytanie z regułami

```
mezczyzna(jan).  
mezczyzna(tomasz).  
mezczyzna(adam).  
mezczyzna(tadeusz).  
mezczyzna(maurycy).  
mezczyzna(antoni).
```

```
rodzic(adam, tadeusz).  
rodzic(adam, antoni).  
rodzic(tadeusz, jan).  
rodzic(jan, tomasz).  
rodzic(antoni, maurycy).
```

  
 $\text{rodzic}(\text{adam}, \text{Z}), \text{mezczyzna}(\text{adam}), \text{ojciec}(\text{Z}, \text{tomasz}).$
 $\text{Z} = \text{tadeusz}$

Arytmetyka w PROLOG-u

Operator	Opis
$X := Y$	Równe
$X \neq Y$	Różne
$X < Y$	X mniejsze od Y
$X > Y$	X większe od Y
$X = < Y$	X mniejsze lub równe Y
$X \geq Y$	X większe lub równe Y
$X + Y$	
$X - Y$	
$X * Y$	
X / Y	
$X // Y$	Dzielenie całkowite
$X \text{ mod } Y$	Reszta z dzielenia

Równość w Prologu

W Prologu istnieje operator porównania `=`, ale jego znaczenie jest trochę odmienne niż w innych językach programowania.

```
?- 1 = 1.  
true
```

```
?- X = Y.  
X = Y.
```

```
?- X = ala, Y = X.  
X = Y, Y = ala.
```

```
?- X = Y, Y = Z, X = ala.  
X = Y, Y = Z, Z = ala.
```

```
?- 3 = (1+2).  
false
```

```
?- X = (1+2).  
X = 1+2.
```

Równość w Prologu

operator `is` wyznacza wartość wyrażenia po prawej stronie.

```
?- X = (1+2).  
X = 1+2.
```

```
?- X is (1+2).  
X = 3.
```

```
?- 3 is 1+2.  
true.
```

```
?- X is Y.  
ERROR: is/2: Arguments are not sufficiently instantiated
```

```
?- X is ala  
ERROR: is/2: Arithmetic: `ala/0' is not a function
```

Arytmetyka w Prologu

```
wlada(rhodri, 874, 878).  
wlada(anarawd, 878, 916).  
wlada(hywel_ap_idwal, 916, 950).  
wlada(lago_ap_idwal, 950, 979).  
wlada(hywel_ap_ieauf, 979, 985).  
wlada(cadwallon, 985, 986).  
wlada(maredudd, 986, 999).  
  
ludnosc(usa, 203).  
ludnosc(indie, 548).  
ludnosc(china, 800).  
ludnosc(brazylia, 108).  
  
obszar(usa, 3).  
obszar(indie, 1).  
obszar(chiny, 4).  
obszar(brazylia, 3).
```

```
ksiaze(X, Y) :-  
    wlada(X, A, B),  
    Y >=A,  
    Y =< B.
```

```
gestosc(X, Y) :-  
    ludnosc(X, A),  
    obszar(X, B),  
    Y is A/B.
```

Rekurencja w PROLOG-u

Rekurencja w prologu polega na wykorzystaniu w regule jej samej.

```
silnia(0,W) :- W is 1.
```

```
silnia(N,W) :- X is N-1,  
            silnia(X, W_N_1),  
            W is N*W_N_1.
```

Listy jako złożony typ danych

- Lista w Prologu przedstawiana jest jako ciąg termów zawartych pomiędzy nawiasami kwadratowymi i oddzielonych przecinkami.
- Nie ma specjalnych funkcji tworzenia list i dostępu do elementów
- W regułach wykorzystuje się notację $[X|Y]$

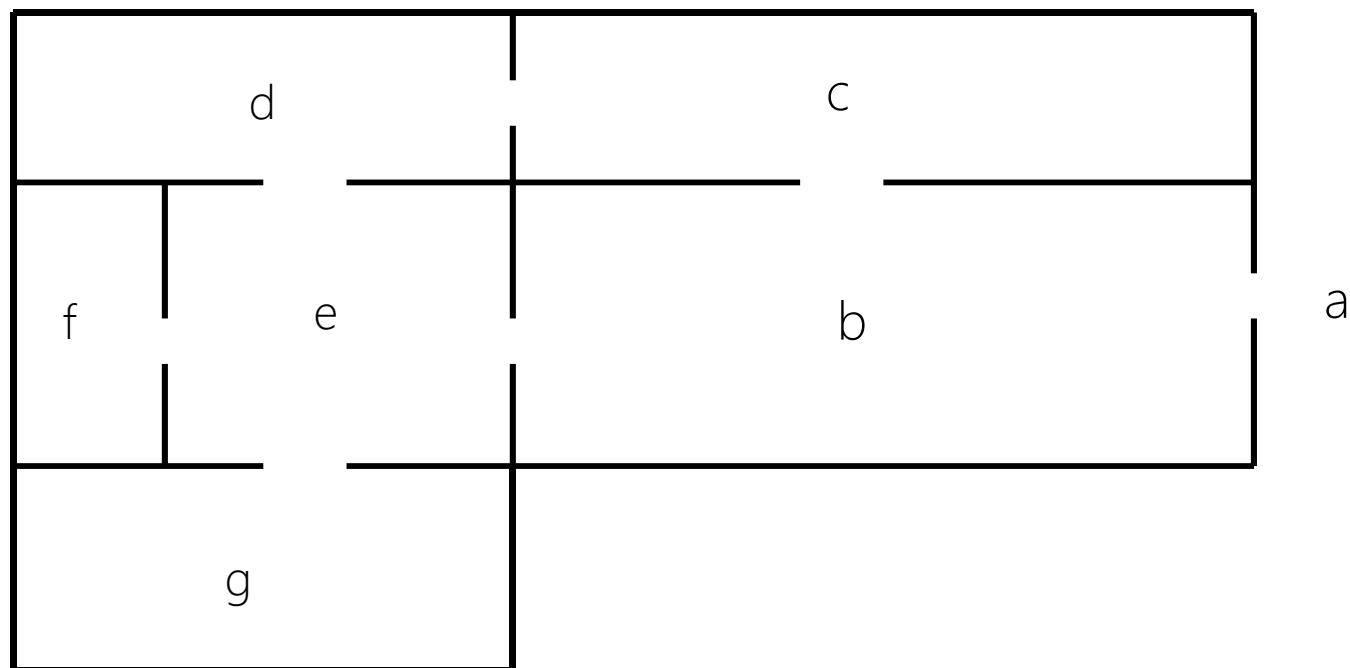
Listy jako złożony typ danych

```
zawiera(Elem, [Elem|_]).  
zawiera(Elem, [_|Reszta]) :- zawiera(Elem, Reszta).
```

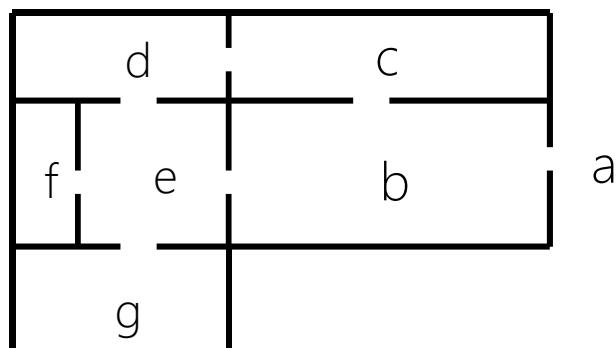
```
polacz([], Lista, Lista).  
polacz([G|Reszta], Lista, [G|Reszta_Wynik]) :-  
    polacz(Reszta, Lista, Reszta_Wynik).
```

```
odwroc([], []).  
odwroc([G|Reszta], Lista) :-  
    odwroc(Reszta, Wynik), polacz(Wynik, [G], Lista).
```

Przykładowy problem z nawracaniem

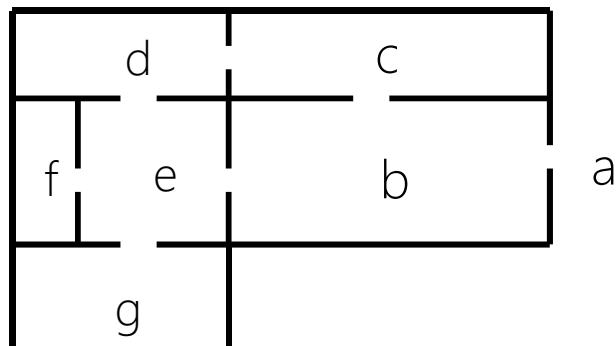


Przykładowy problem z nawracaniem



przejscie(a,b).
przejscie(b,e).
przejscie(b,c).
przejscie(d,e).
przejscie(c,d).
przejscie(e,f).
przejscie(g,e).

Przykładowy problem z nawracaniem

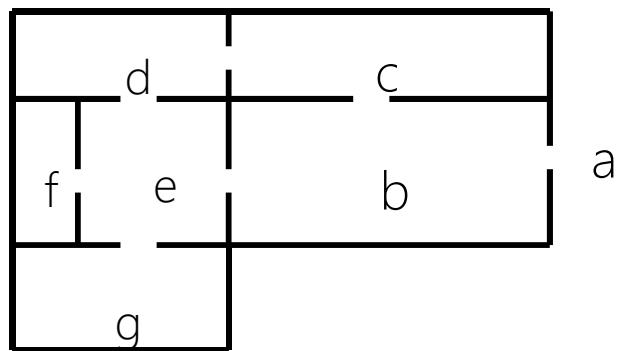


droga(X,X).

droga(X,Y) :- przejscie(X,Z), droga(Z,Y).

droga(X,Y) :- przejscie(Z,X), droga(Z,Y).

Przykładowy problem z nawracaniem



`droga(X,X,_).`

`droga(X,Y,T) :- przejscie(X,Z),
 \+ member(Z,T),
 droga(Z,Y,[Z|T]).`

`droga(X,Y,T) :- przejscie(Z,X),
 \+ member(Z,T),
 droga(Z,Y,[Z|T]).`

JĘZYKI DOMENOWE I FLUENT INTERFACES

„Programy są przeznaczone do czytania przez ludzi, a tylko przypadkowo mogą być wykonywane przez komputer”

Donald Knuth

Języki domenowe

Języki domenowe (ang. Domain Specific Languages)
– języki programowania do rozwiązywania konkretnego problemu

Ograniczony zakres

Rozwiązuje konkretny problem domenowy w bardzo dobry sposób

Języki domenowe



Który fragment jest bardziej czytelny?

```
var results =  
collection  
    .Where(i=>i.Age > 18)  
    .OrderBy(i=>i.FirstName)  
    .ThenBy(i=>i.LastName)  
    .ToList()
```

```
var results = new List<Person>()  
  
foreach(var item in collection) {  
    if(item.Age > 18)  
        results.Add(item);  
}  
  
results.Sort(ComparePersons);
```

Języki domenowe - zewnętrzne

Języki domenowe zewnętrzne są niezależnymi językami posiadającymi własną składnię i semantykę.

Przykładami są: CSS, SQL

Języki domenowe - wewnętrzne

Języki domenowe wewnętrzne (osadzone) są to specjalnie przygotowane API w określonym języku ogólnego przeznaczenia wykorzystujące jego składnię i semantykę. Często nazywane Fluent API

Fluent API

„Fluent Interface jest metodą dla konstruowania API aplikacji, w taki sposób, aby czytelność kodu była jak najbardziej zbliżona do języka naturalnego.”

Martin Fowler

Fluent API w języku C#

- Metody rozszerzające
- Łącznie metod (method chaining)
- Fluent interfaces

Metody rozszerzające, a DSL

```
public void Metoda() {  
    TimeSpan jednaGodzina = new TimeSpan(1, 0, 0);  
}
```

Metody rozszerzające, a DSL

```
static class Extensions {
    public static TimeSpan Godzina(this int value) {
        return new TimeSpan(value, 0, 0);
    }
}

public void Metoda() {
    TimeSpan jednaGodzina = 1.Godzina();
}
```

Łączenie metod (method chaining)

Pozwala na wielokrotne wywoływanie różnych metod bez konieczności wykorzystania zmiennych do przechowywania wyników pośrednich.

Łączenie metod

```
class Bus {  
    List<Passenger> _passengers;  
    List<Beacon> _beacons;  
  
    public void AddPassenger(Passenger p) {  
        _passengers.Add(p);  
    }  
  
    public void SetBeacon(Beacon b) {  
        _beacons.Add(b);  
    }  
    ...  
}
```

Łączenie metod

```
var bus = new Bus();
bus.AddPassenger(new Passenger());
bus.AddPassenger(new Passenger());
bus.SetBeacon(new Beacon());
bus.SetBeacon(new Beacon());
bus.CloseDoor();
bus.TurnOnBeacons();
```

Łączenie metod

```
class Bus {  
    List<Passenger> _passengers;  
    List<Beacon> _beacons;  
  
    public Bus AddPassenger(Passenger p) {  
        _passengers.Add(p);  
        return this;  
    }  
  
    public Bus SetBeacon(Beacon b) {  
        _beacons.Add(b);  
        return this;  
    }  
}
```

Łączenie metod

```
var bus =  
    new Bus()  
    .AddPassenger(new Passenger())  
    .AddPassenger(new Passenger())  
    .SetBeacon(new Beacon())  
    .SetBeacon(new Beacon())  
    .CloseDoor()  
    .TurnOnBeacons();
```

Łączenie metod

```
var results =  
    collection  
        .where(i=>i.Age > 18)  
        .orderBy(i=>i.FirstName)  
        .thenBy(i=>i.LastName)
```

Łączenie metod w jQuery

```
$ (function () {  
    $('#lista') .  
        .find('> li')  
  
        .filter(':first').addClass('specjalny').end()  
        .find('ul')  
            .css('border', '1px solid red')  
  
        .find('li:last').addClass('specjalny').end()  
            .end()  
        .end()  
        .find('li')  
            .append('każdy li');  
}) ;
```

Kontekst

Kontekst jest to obiekt, który ułatwia łączenie metod oraz zachowuje informacje z poprzednich wywołań

Proste wywołanie

Proste wywołanie, jest to wywołanie, które zwraca kontekst tego samego typu co obiekt na którym zostało wykonane

Proste wywołanie

```
var bus =  
    new Bus()  
        .AddPassenger(new Passenger())  
        .AddPassenger(new Passenger())  
        .SetBeacon(new Beacon())  
        .SetBeacon(new Beacon())  
        .CloseDoor()  
        .TurnOnBeacons();
```

AddPassenger
SetBeacon
CloseDoor
TurnOnBeacons



Proste wywołanie

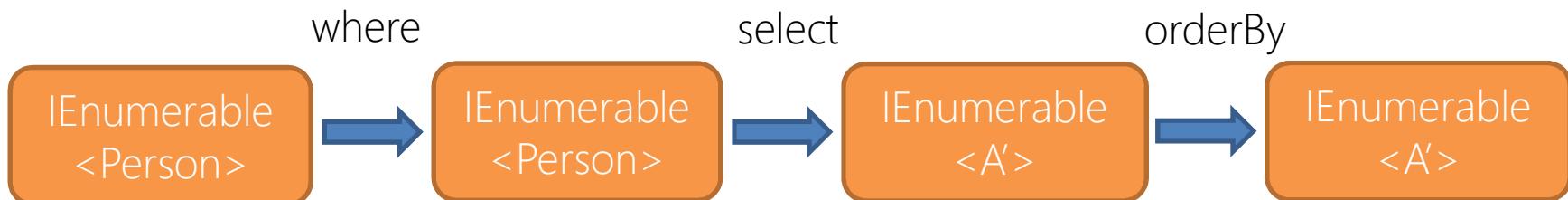
```
var results =  
    collection  
        .where(i=>i.Age > 18)  
        .orderBy(i=>i.FirstName)  
        .thenBy(i=>i.LastName)
```

Złożone wywołanie

Złożone wywołanie, jest to wywołanie, które zwraca kontekst innego typu co obiekt na którym zostało wykonane

Złożone wywołanie

```
var results =  
collection  
.where(i=>i.Age > 18)  
.select(i=> new { Imie = i.FirstName, Nazwisko = i.LastName })  
.orderBy(p=>p.Imie);
```



Autobus ponownie

```
var bus =  
    new Bus()  
        .AddPassenger(new Passenger())  
        .AddPassenger(new Passenger())  
        .SetBeacon(new Beacon())  
        .SetBeacon(new Beacon())  
        .CloseDoor()  
        .TurnOnBeacons();
```

A co jeśli będziemy chcieli mieć pewność, że w autobusie na pewno ustawimy co najmniej jednego pasażera lub co najmniej jeden beacon?

Autobus ponownie

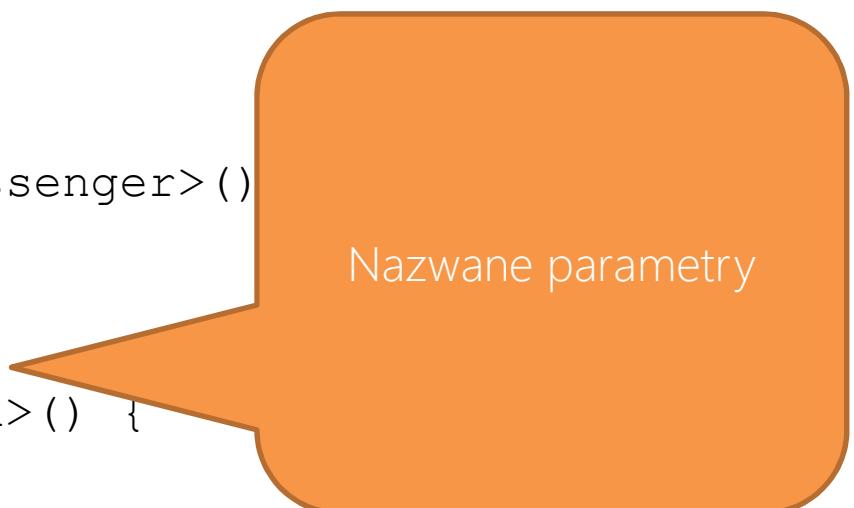
```
var bus =  
    new Bus(  
        new List<Passenger>() {  
            new Passenger(),  
            new Passenger(),  
        },  
        new List<Beacon>() {  
            new Beacon(),  
            new Beacon()  
        })  
.CloseDoor()  
.TurnOnBeacons();
```



Kod poprawny, ale czy ładny?

Autobus ponownie – nazwane parametry

```
var bus =  
    new Bus(  
        passengers: new List<Passenger>()  
            new Passenger(),  
            new Passenger(),  
        ),  
        beacons: new List<Beacon>() {  
            new Beacon(),  
            new Beacon()  
        } )  
.CloseDoor()  
.TurnOnBeacons();
```



Nazwane parametry

Autobus ponownie - fluent

```
class Bus {  
    ...  
    protected Bus() { ... }  
    public static Bus WithPassenger(Passenger p) {  
        var bus = new Bus();  
        bus.AddPassenger(p);  
        return bus;  
    }  
    public static Bus WithBeacon(Beacon b) {  
        var bus = new Bus();  
        bus.SetBeacon(b);  
        return bus;  
    }  
}
```

Autobus ponownie - fluent

```
var bus = new Bus();
```

Błąd. Konstruktor jest protected.

```
var bus = Bus.WithBeacon(new Beacon())
    .SetBeacon(new Beacon())
    .AddPassenger(new Passenger());
```

Builder

```
class BusBuilder {  
  
    public BusBuilder(Bus bus) { _bus = bus; }  
  
    public BusBuilder SetBeacon(Beacon b)  
    { _bus.SetBeacon(b); return this; }  
  
    public BusBuilder AddPassenger(Passenger p)  
    { _bus.AddPassenger(p); return this; }  
  
    public Bus Get() {  
        return _bus;  
    }  
}
```

Builder

```
class Bus {  
    ...  
    protected Bus() { ... }  
  
    public static BusBuilder Create() {  
        var bus = new Bus();  
        return new BusBuilder(bus);  
    }  
}
```

Builder

```
Bus.Create()
    .AddPassenger(new Passenger())
    .SetBeacon(new Beacon())
    .AddPassenger(new Passenger())
    .SetBeacon(new Beacon())
    .Get();
```

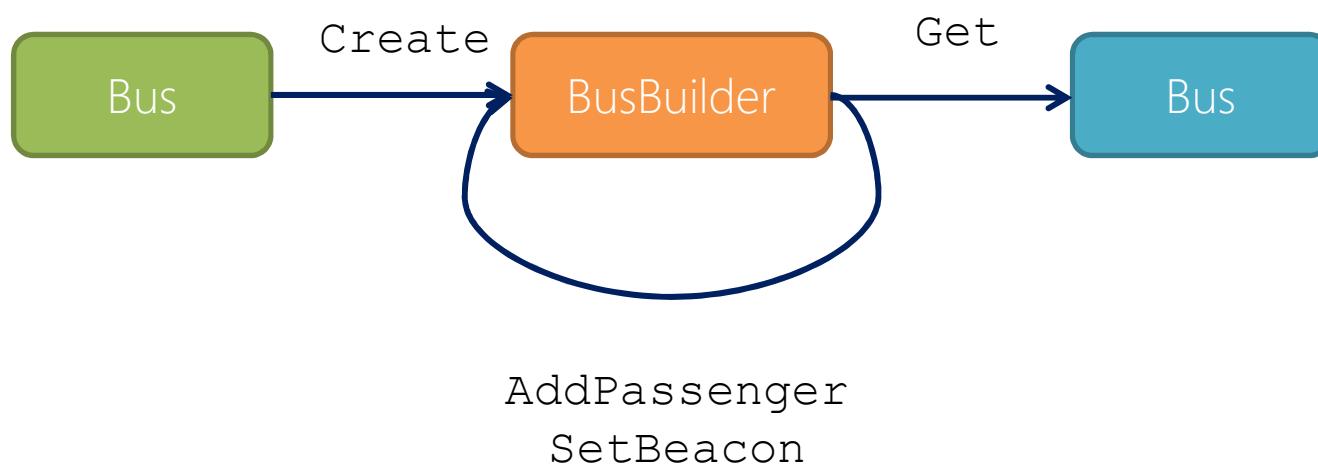
Builder

```
class BusBuilder {  
    ...  
    public Bus Get => _bus;  
}  
  
class Bus {  
    ...  
    public static BusBuilder Create =>  
        new BusBuilder(new Bus());  
}
```

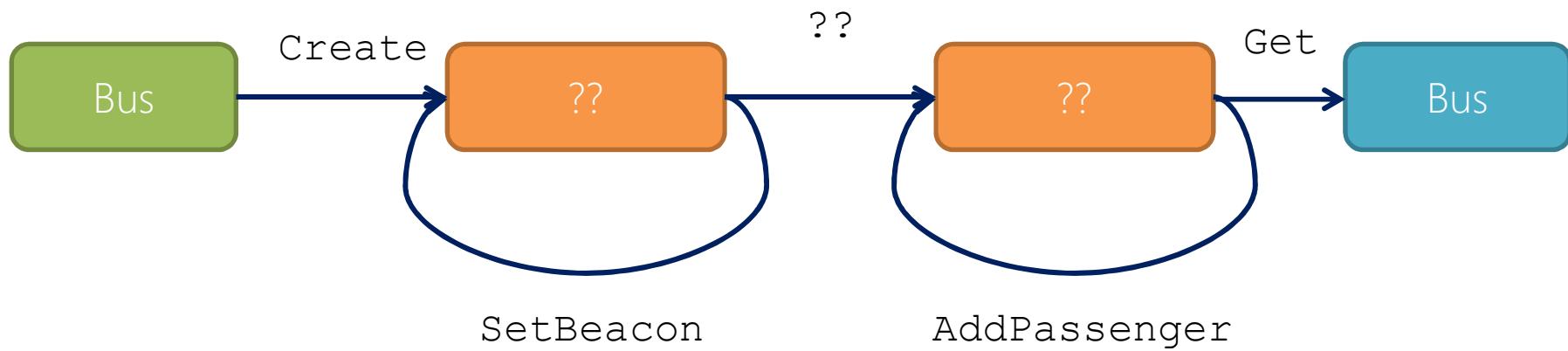
Builder

```
Bus.Create  
    .AddPassenger(new Passenger())  
    .SetBeacon(new Beacon())  
    .AddPassenger(new Passenger())  
    .SetBeacon(new Beacon())  
    .Get;
```

Builder



Builder



Fluent interfaces

Wykorzystanie interfejsów do stworzenia Fluent API

Fluent interfaces

```
interface IBusBuilder_Beacons
{
    IBusBuilder_Beacons SetBeacon(Beacon b);
    IBusBuilder_Passengers AddPassenger(Passenger p);
}

interface IBusBuilder_Passengers
{
    IBusBuilder_Passengers AddPassenger(Passenger p);
    Bus Get { get; }
}
```

Fluent interfaces

```
class BusBuilder : IBusBuilder_Passengers, IBusBuilder_Beacons
{
    private Bus _bus;

    public BusBuilder(Bus bus) { _bus = bus; }

    public IBusBuilder_Beacons SetBeacon(Beacon b)
    { _bus.SetBeacon(b); return this; }

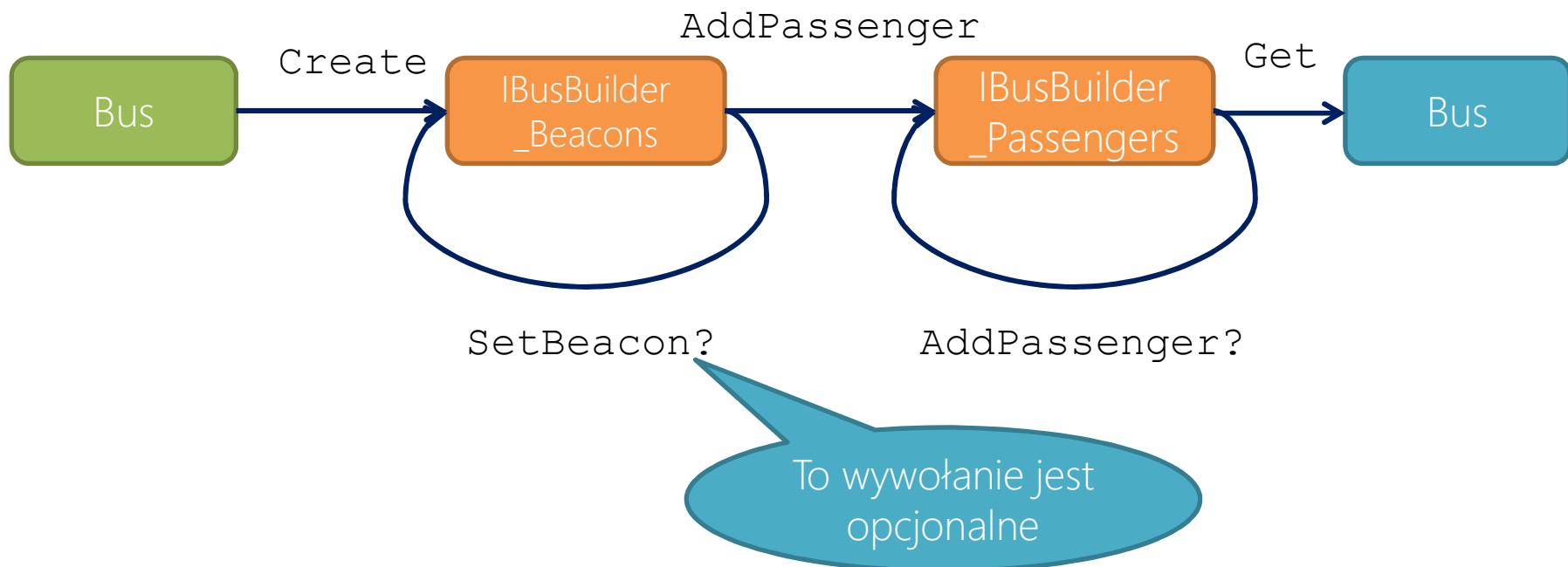
    public IBusBuilder_Passengers AddPassenger(Passenger p)
    { _bus.AddPassenger(p); return this; }

    public Bus Get => _bus;
}
```

Fluent interfaces

```
class Bus {  
    ...  
    public static IBusBuilder_Beacons Create  
        => new BusBuilder(new Bus());  
}
```

Fluent interfaces



Fluent interfaces

```
interface IBusBuilder_Beacons1
{
    IBusBuilder_Beacons2 SetBeacon(Beacon b);
}

interface IBusBuilder_Beacons2
{
    IBusBuilder_Beacons2 SetBeacon(Beacon b);
    IBusBuilder_Passengers AddPassenger(Passenger p);
}
```

Fluent interfaces

```
class BusBuilder
    : IBusBuilder_Passengers, IBusBuilder_Beacons1, IBusBuilder_Beacons2
{
    private Bus _bus;

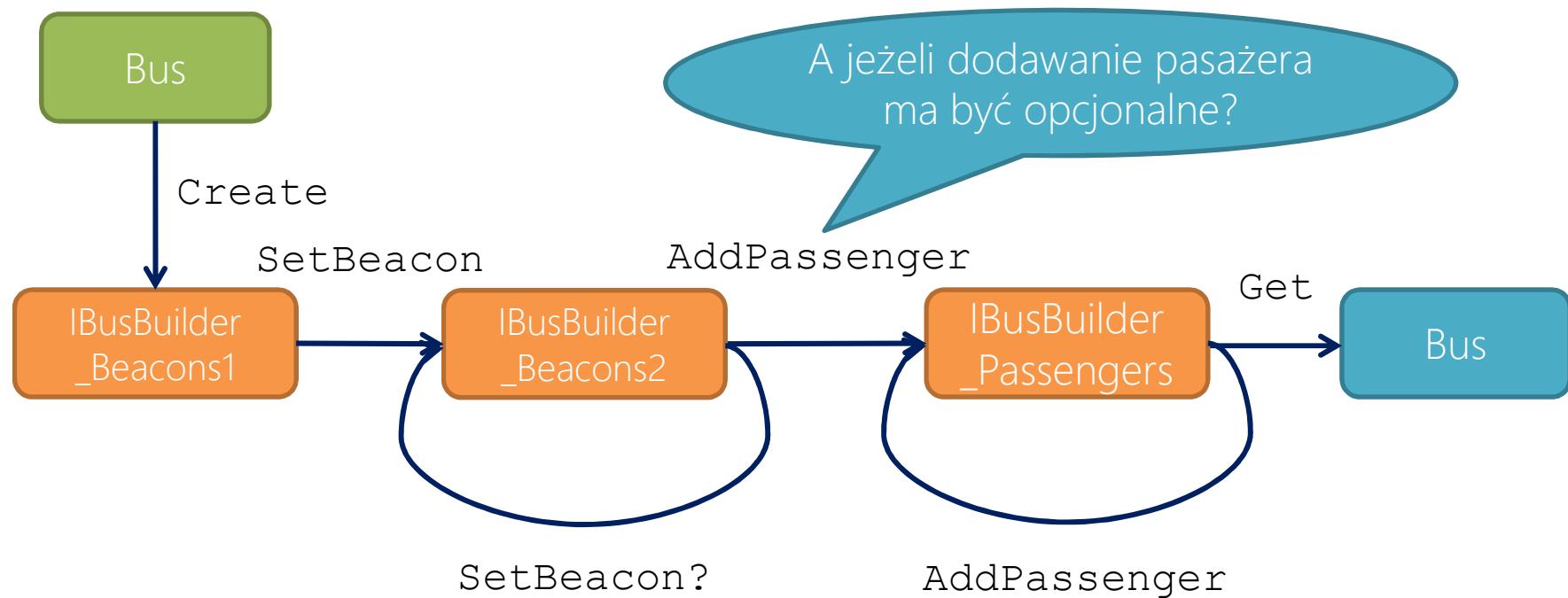
    public BusBuilder(Bus bus) { _bus = bus; }

    public IBusBuilder_Beacons2 SetBeacon(Beacon b)
    { _bus.SetBeacon(b); return this; }

    public IBusBuilder_Passengers AddPassenger(Passenger p)
    { _bus.AddPassenger(p); return this; }

    public Bus Get => _bus;
}
```

Fluent interfaces

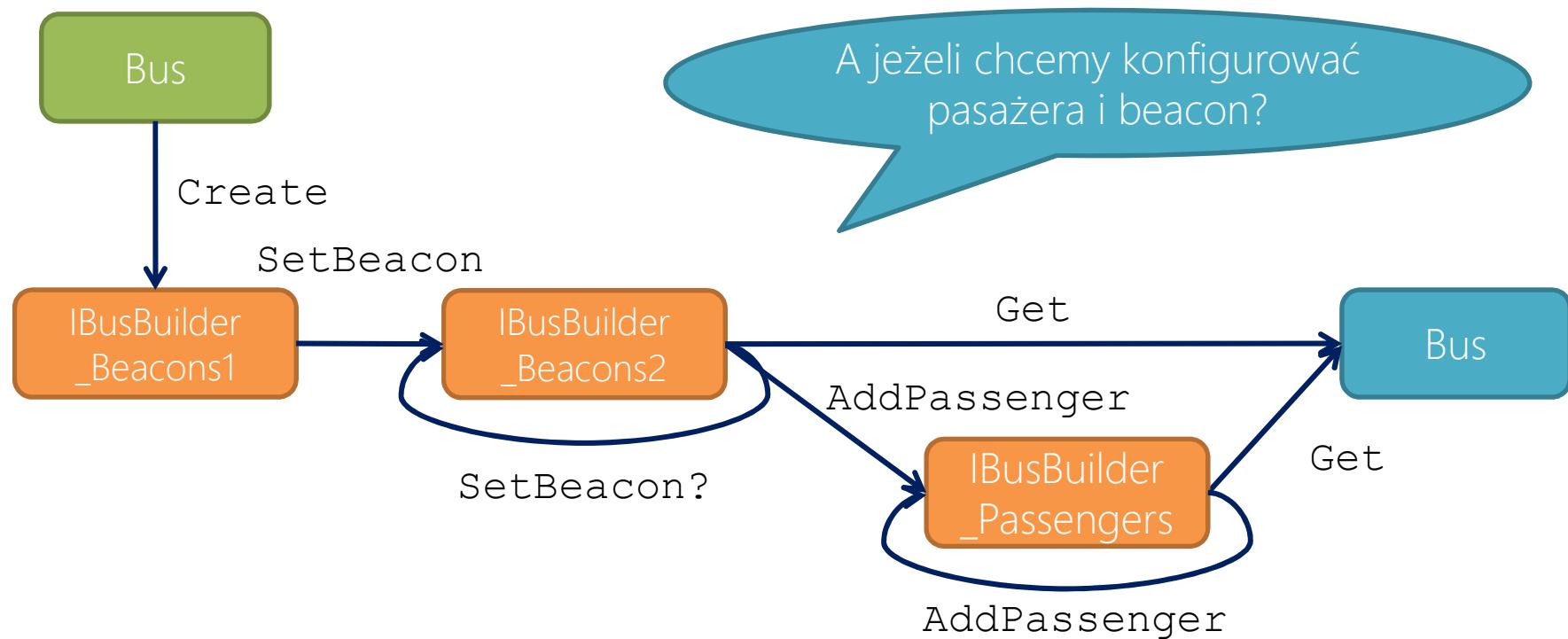


Fluent interfaces

```
interface IBusBuilder_Beacons1
{
    IBusBuilder_Beacons2 SetBeacon(Beacon b);
}

interface IBusBuilder_Beacons2
{
    IBusBuilder_Beacons2 SetBeacon(Beacon b);
    IBusBuilder_Passengers AddPassenger(Passenger p);
    Bus Get { get; }
}
```

Fluent interfaces



Fluent interfaces

