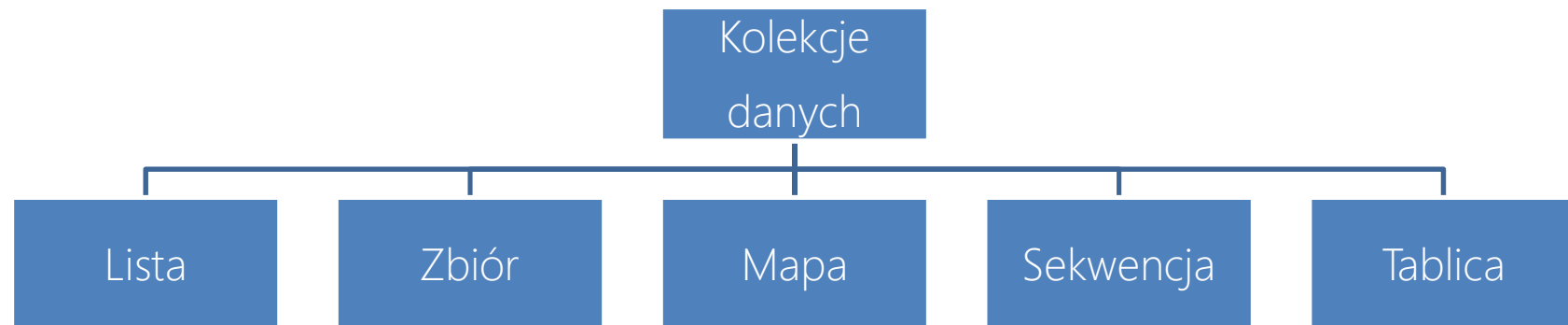
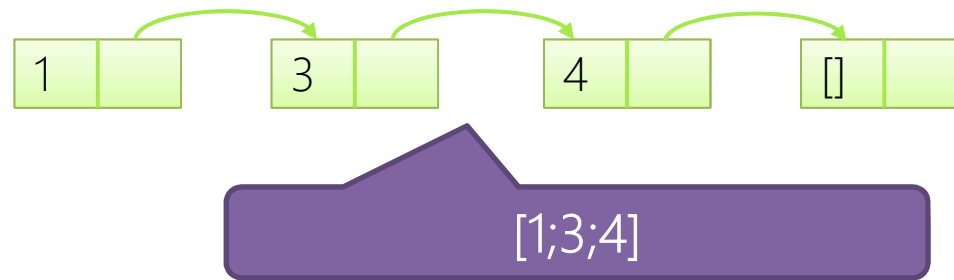


KOLEKCJE DANYCH W F#



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`



`[1;3;4]`

`1::3::4::[]`

Te zapisy stworzą listy ekwiwalentne

`1::3::4`

Zapis błędny

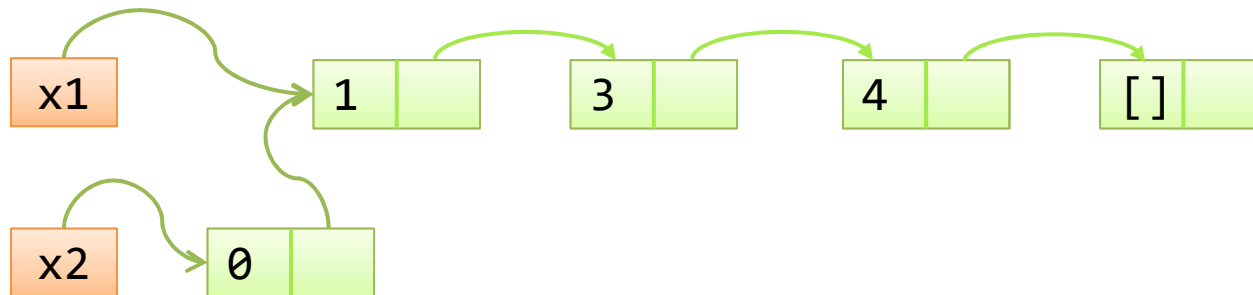
- Listy są niemodyfikowalne. Operator `::` tworzy nową listę zamiast modyfikować istniejącą

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

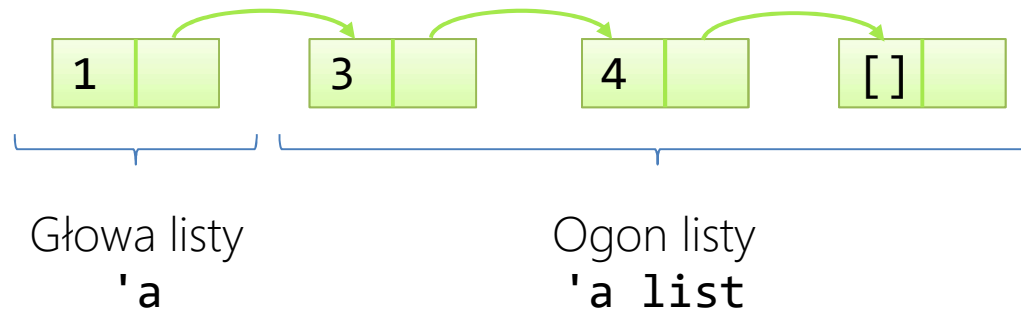
Skracanie list (2)

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



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

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;;
```



Mapy

Niemodyfikowalny słownik, w którym elementy dostępne są poprzez klucz

```
let mapa = Map.ofList [("Ala", 3);("Tomek", 5);("Adam", 4)]
```

```
mapa.[ "Ala" ]
```

Sekwencje

Sekwencja to leniwa kolekcja obiektów tego samego typu

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

```
let lista = [for n in 1I..20000I do yield (n, silnia n)]  
List.iter (fun (n,v)->printfn "%A %A" n v) lista
```

Sekwencje

Sekwencja to leniwa kolekcja obiektów tego samego typu

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

```
let sekwencja = seq {for n in 1I..20000I do yield (n,silnia n)}  
Seq.iter (fun (n,v)->printfn "%A %A" n v) sekwencja
```

Sekwencje

```
let dane = [  
  {imie="Cezary"; nazwisko="Adamski"; wiek=20; rokStudiow=2};  
  {imie="Franek"; nazwisko="Relke"; wiek=20; rokStudiow=2};  
  {imie="Ala"; nazwisko="Kot"; wiek=19; rokStudiow=1};  
  {imie="Ewa"; nazwisko="Nowacka"; wiek=22; rokStudiow=2};  
  {imie="Adam"; nazwisko="Kowalski"; wiek=21; rokStudiow=3};  
]
```

```
let dane = List.filter (fun o->o.rokStudiow = 2) dane  
let dane = List.sortBy (fun o->o.nazwisko) dane  
let wynik = List.map (fun o->{|imie=o.imie;  
                                nazwisko=o.nazwisko|}) dane  
List.iter (fun o->printfn "%A" o) wynik
```


Sekwencje

```
let dane = [  
    {imie="Cezary"; nazwisko="Adamski"; wiek=20; rokStudiow=2};  
    {imie="Franek"; nazwisko="Relke"; wiek=20; rokStudiow=2};  
    {imie="Ala"; nazwisko="Kot"; wiek=19; rokStudiow=1};  
    {imie="Ewa"; nazwisko="Nowacka"; wiek=22; rokStudiow=2};  
    {imie="Adam"; nazwisko="Kowalski"; wiek=21; rokStudiow=3};  
]
```

```
let dane = Seq.filter (fun o->o.rokStudiow = 2) dane  
let dane = Seq.sortBy (fun o->o.nazwisko) dane  
let dane = Seq.map (fun o->{|imie=o.imie;  
                                nazwisko=o.nazwisko|}) dane  
let wynik = Seq.toList dane  
List.iter (fun o->printfn "%A" o) wynik
```