// Lab 1

// Маневич Егор

let list = [ 0..5 ]

let mutable result = Seq.empty

let print (x:'a) =

printfn "%A" x

x

let printIgnore (x:'a) = printfn "%A" x

// Exercise 1 - 1 Double each item of a list

// Using map

let mutable temp = []

list |> (Seq.map (fun x -> temp <- x :: x :: temp) >> Seq.toList) |> ignore //eaeger execution to initialize temp

result <- Seq.rev temp //lazy reverse

// Using fold

result <- list |> (Seq.fold (fun state x -> x :: x :: state) [] >> Seq.rev) //fully lazy

// Exercise 1 - 2 Substract items of two lists

let list2 = List.rev [ -5..3 ]

//print list

//print list2

result <- Seq.map2 (-) list list2

// Exercise 2 - 1 Filter even

let isEven x = x % 2 = 0

result <- Seq.filter isEven list

// Exercise 2 - 2 Filter x's that are not 10 ≤ x ≤ 100.

let l = [ -999; -55; 0; 1; 9; 10; 12; 44; 55; 99; 100; 101; 999 ]

let isInRange x = not (10 <= x && x <= 100)

result <- Seq.filter isInRange l

// Exercise 3 - 1 Square each item of a list, then sums them and then makes square root of it

let show (x:'a) =

// print x |> ignore

x

let showIgnore (x:'a) =

// printIgnore x |> ignore

()

let sqr x = int ((double x) \*\* 2.0)

let sqrSeq = Seq.map sqr

let squareSeqThanApplyFuncThanSquare func = sqrSeq >> show >> func >> show >> sqr >> showIgnore

//Using Seq.sum

list |> squareSeqThanApplyFuncThanSquare Seq.sum

//Using reduce

let sumSeq = Seq.reduce (+)

list |> squareSeqThanApplyFuncThanSquare sumSeq

//Using fold

let sumSeq2 = Seq.fold (+) 0

list |> squareSeqThanApplyFuncThanSquare sumSeq2

// Exercise 4 - 1 Sort by magnitude of sin(x) ascending

let pi = (double Math.PI)

let a = [ pi; 4.0\*pi; 0.0; 3.0/2.0\*pi; pi/2.0; 2.0\*pi; ]

//printIgnore a //[3.141592654; 12.56637061; 0.0; 4.71238898; 1.570796327; 6.283185307]

//let b = List.map sin a

//printIgnore b //[1.224606354e-16; -4.898425415e-16; 0.0; -1.0; 1.0; -2.449212708e-16]

let c = List.sortBy sin a

//printIgnore c //[4.71238898; 12.56637061; 6.283185307; 0.0; 3.141592654; 1.570796327]

// Exercise 4 - 2 Sort by length of lists descending

let listOfArrays = [[|1..3|]; [|1..2|]; [|1..6|]; [|1..4|]; [|1..3|]]

let getLength (x: 'a array) = x.Length

let sorted = List.sortByDescending getLength listOfArrays

//sorted |> printIgnore //[[|1; 2; 3; 4; 5; 6|]; [|1; 2; 3; 4|]; [|1; 2; 3|]; [|1; 2; 3|]; [|1; 2|]]

// Exercise 6 - 1 member-if function

let rec memberIf proc ls =

match ls with

| [] -> []

| head :: tail -> if proc head then ls else memberIf proc tail

let testMemberIf l = memberIf (fun x -> x > 0) l |> printIgnore

//testMemberIf [0;0;0;0]

//testMemberIf [-1;-4;0;-6]

//testMemberIf [-1;-4;1;2;5]

//testMemberIf [-1;-4;1;-2;-5]

// Exercise 6 - 2 map function

let rec map func args =

match args with

| [] -> []

| firstList :: argsRest ->

match argsRest with

| [] -> firstList

| secondList :: restOfargsRest ->

let result = List.map2 func firstList secondList

let newArgs = result :: restOfargsRest

map func newArgs

let testMap l = map (+) l |> printIgnore

//testMap []

//testMap [[1]]

//testMap [[1]; [1]]

//testMap [[]; []; []]

//testMap [[1; 2; 3]; [1; 2; 3]; [1; 2; 3]; [1; 2; 3];]

//testMap [[1; 2; 3]; [1]; [1; 2];] //throws exception

result |> Seq.toList |> printIgnore