Programming 2
Tutorial 4

Functional Programming (2)

Nedim Šišić

2023

Exercises

- Write a function which checks if the elements of a 2-tuple are the same
- Write a (non-recursive) function which in a single call checks if the first and third element of a list are the same

Exercises

- Write a function which checks if the elements of a 2-tuple are the same

```
let same x =
  match x with
   | (a, b) when (a = b) -> true
   | _ -> false;;
```

- Write a (non-recursive) function which in a single call checks if the first and third element of a list are the same

```
let firstEqualThird x =
  match x with
    | h1::(_::(h3::_)) when (h1 = h3) -> true
    | _ -> false;;
```

Exercises

Write a function which “compresses” a list by removing consecutive copies

`compress [1; 1; 1; 2; 2; 1]` → `[1; 2; 1]`

We can simplify our code by using the “as” keyword

```
let rec compress l =
  match l with
    | a::(b::_ as tail) when a = b -> compress tail
    | a::(b::_ as tail) -> a::compress tail
    | remainder -> remainder;;
```

Exercises

- Write a function that takes a list *l* and an integer *n*, and returns a list containing all the elements in *l* greater than *n*.
- Write a function that takes a list and an integer *n* and returns the *n*-th element of the list. If *n* is greater than the length of the list, the function returns 0.

Exercises

- Write a function that takes a list *l* and an integer *n*, and returns a list containing all the elements in *l* greater than *n*.

```
let rec getGreaterThan l n =
  match l with
    | h::t when (h > n) -> h :: getGreaterThan h t
    | h::t -> greaterThan t n
    | [] -> [];;
```

Exercises

- Write a function that takes a list and an integer *n* and returns the *n*-th element of the list. If *n* is greater than the length of the list, the function returns 0.

```
let rec nthElement l n =
  match l with
    | [] -> 0
    | _ when (n <= 0) -> 0
    | h::t when (n = 1) -> h
    | h::t -> nthElement t (n - 1);;
```

Higher order functions

- Write a function that takes a function *f* and value *v,* and returns *f (f (v))*.
- Write a function that takes a function *f* and a list *l,* and applies *f* to each element in *l*.

Higher order functions

- Write a function that takes a function *f* and value *v,* and returns *f (f (v))*.

```
let applyTwice f v =
  f (f v);;
```

- Write a function that takes a function *f* and a list *l,* and applies *f* to each element in *l*.

```
let rec myMap f l =
  match l with
    | [] -> []
    | h::t -> f h :: myMap f t;;
```

Exercises

- Write a function that multiplies each element of a list with a given number n.
- Write a function that takes a list *l* and an integer *n*, and returns a list containing all the elements in *l* greater than *n*.
- Use both fold functions separately to apply logical implication to a given list. Apply both functions on the list [true;true;true] and an initial value of false.

Exercises

- Write a function that multiplies each element of a list with a given number *n*.

```
let multAll l n =
  List.map (fun x -> x*n) l;;
```

- Write a function that takes a list *l* and an integer *n*, and returns a list containing all the elements in *l* greater than *n*.

```
let greaterThan l n =
  List.filter (fun x -> x > n) l;;
```

Exercises

- Use both fold functions separately to apply logical implication to a given list. Apply both functions on the list [true;true;true] and an initial value of false.

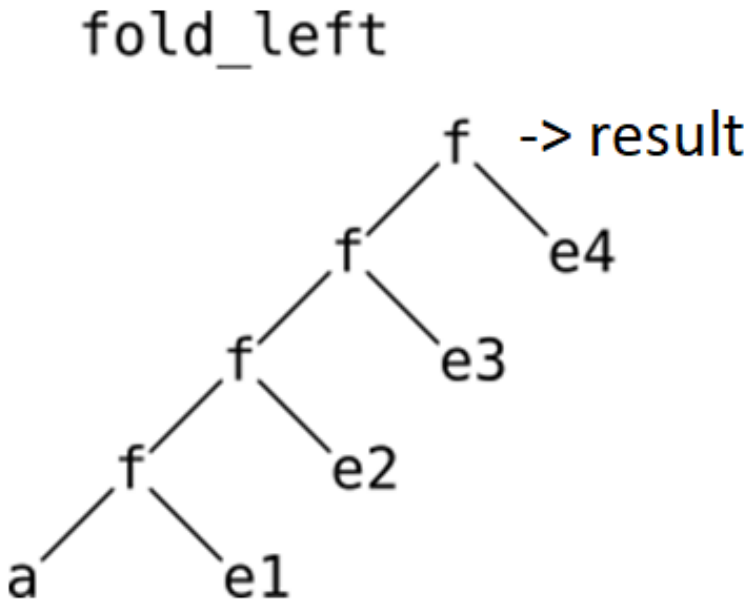
```
let implies a b =
  match (a,b) with
    | (true,false) -> false;
    | _ -> true;;

List.fold_left implies false [true; true; true];;

List.fold_right implies [true; true; true] false;;
```

Lists – folding left and folding right

List.fold_left f a [e1; e2; e3; e4];;



List.fold_right f [e1; e2; e3; e4] a;;