FPV Tutorübung

Woche 7

OCaml: List-Module 2, Mappings, Operator Functions

Manuel Lerchner

08.06.2023

T01: List Module Part 2

- Use functions from the List-Module!

Implement the following functions without defining any recursive functions yourself:

1. ⊗ **float_list** 0 von 1 Tests bestanden
   Implement the function `float_list : int list -> float list` that converts all ints in the list to floats.

2. ⊗ **to_string** 0 von 1 Tests bestanden
   Implement the function `to_string : int list -> string` that builds a string representation of the given list. E.g.: `"[0;42;123;420;1;]"`

3. ⊗ **part_even** 0 von 1 Tests bestanden
   Implement the function `part_even : int list -> int list` that partitions all even values to the front of the list.

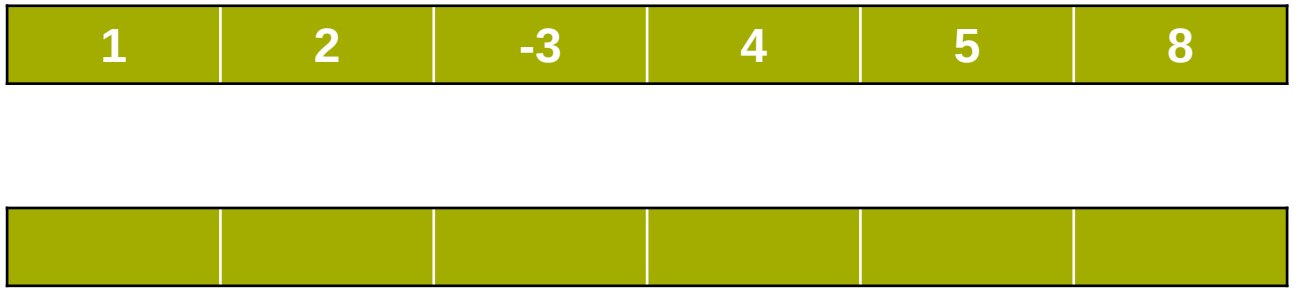
4. ⊗ **squaresum** 0 von 1 Tests bestanden
   Implement the function `squaresum : int list -> int` that computes $\sum_{i=1}^{n} x_i^2$ for a list $[x_1, \ldots, x_n]$.

T01: List Module Part 2

- Selected Functions from the List-Module
  - `List.map ('a -> 'b) -> 'a list -> 'b list`
  - `List.fold_left ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`
  - `List.find_opt ('a -> bool) -> 'a list -> 'a option`
  - `List.filter ('a -> bool) -> 'a list -> 'a list`

T01: List Module Part 2

- `List.map ('a -> 'b) -> 'a list -> 'b list`



T01: List Module Part 2

- `List.fold_left  ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`

T01: List Module Part 2

- `List.find_opt` `('a -> bool) -> 'a list -> 'a option`

T01: List Module Part 2

- `List.filter` `('a -> bool) -> 'a list -> 'a list`

T02: Mappings

Idea: Create a Dictionary-Datastructure

```
age_dictionary = {
    "John": 25,
    "Mary": 20,
    "Tom": 30
}
```

1. Implement these functions to work with mappings based on associative lists:
   1. ⊗ **is_empty** 0 of 1 tests passing
      `is_empty : ('k * 'v) list -> bool`
   2. ⊗ **get** 0 of 1 tests passing
      `get : 'k -> ('k * 'v) list -> 'v option`

      If the key is mapped to multiple values, return the first such value
   3. ⊗ **put** 0 of 1 tests passing
      `put : 'k -> 'v -> ('k * 'v) list -> ('k * 'v) list`

      If the key is already mapped to one or more values, remove those pairs first
   4. ⊗ **contains_key** 0 of 1 tests passing
      `contains_key : 'k -> ('k * 'v) list -> bool`
   5. ⊗ **remove** 0 of 1 tests passing
      `remove : 'k -> ('k * 'v) list -> ('k * 'v) list`

      If the key is mapped to multiple values, remove all such values
   6. ⊗ **keys** 0 of 1 tests passing
      `keys : ('k * 'v) list -> 'k list`
   7. ⊗ **values** 0 of 1 tests passing
      `values : ('k * 'v) list -> 'v list`

T02: Mappings

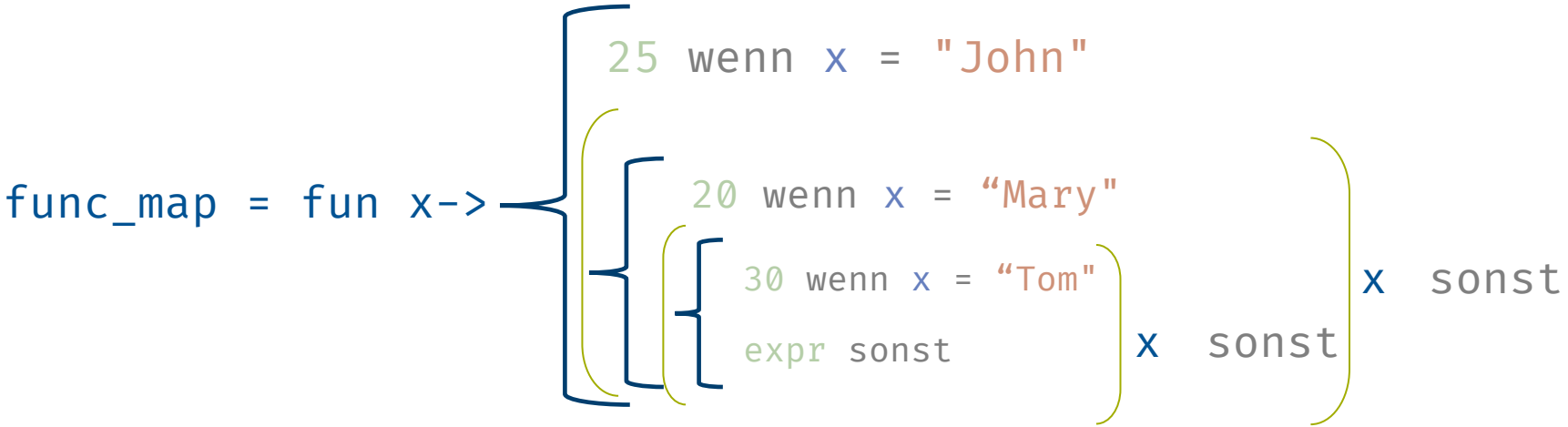
- How to store dictionaries?
  - Association Lists
  - Functional mapping

```
assoc_list = [
    ("John", 25),
    ("Mary", 20),
    ("Tom", 30)
]
```

```
func_map = fun x->
    25 wenn x = "John"
    20 wenn x = “Mary"
    30 wenn x = “Tom"
    expr sonst
```

T02: Functional Mappings

- Every layer saves **exactly** one datapoint
  - If the parameter matches the datapoint -> return its value
  - Else delegate to sub-function



T03: Operator Functions

In OCaml, infix notation of operators is just syntactic sugar for a call to the corresponding function. For example, the binary addition `+` merely calls the function `(+) : int -> int -> int`.

1. Discuss why this is a very useful feature.

Note: This is a tutorial exercise, you do not need to submit anything for this exercise.