OCaml: Practice

Concepts of Programming Languages
Lecture 3

CAS CS 320

Announcements

» Assignment 1 is due on Thursday, 8pm (can submit until 11:59pm)

» Please sign the course manual

» Office Hours have started! Look at calendar on course webpage

» Update standard library:

- `cd path/to/stdlib320`
- `git pull`
- `eval $(opam env)`
- `opam upgrade stdlib320`

Outline

» OCaml practice today!

» We will cover the following:

- Converting loops to recursion
- How to get around mutating variables
- Nested Loops
- Lists

» Leetcode problems!

demo

(loops and recursion)

Iteration vs Recursion

```
int fact = 1;
int n = 10;
int x = 5;
int y = 2;
for (i = 1 ; i < n+1 ; i++) {
  for (j = i+1; j < n+1; j++) {
    x = x + i;
    y = x + y + i;
    fact = fact * i;
  }
}
```

```
let rec f i j fact x y n =
  if i = n+1 then (fact, x, y)
  else if j = n+1 then f (i+1) (i+2) fact x y n
  else
    let x_new = x + i in
    let y_new = x_new + y + i in
    let fact_new = fact * i in
    f i (j+1) fact_new x_new y_new n
```

Lists

» Very much like the linked list, not vectors

» Syntax:

- `[]` for an empty list
- `x::l` for “cons”-ing an element `x` to the front of list `l`
- `[`x_1`;` x_2`; …;` x_n`]` for a fixed list

Some Examples

'a list
```
let l1 = []
```

int list
```
let l2 = 1::l1
```

int list
```
let l3 = 2::3::l2
```

int list
```
let l4 = [1; 2; 3]

(* Are l3 and l4 equal? *)
```

Some Examples

'a list
```
let l1 = []
```

int list
```
let l2 = 1::l1
```

int list
```
let l3 = 2::3::l2
```

int list
```
let l4 = [1; 2; 3]
```

```
(* Are l3 and l4 equal? *)
```

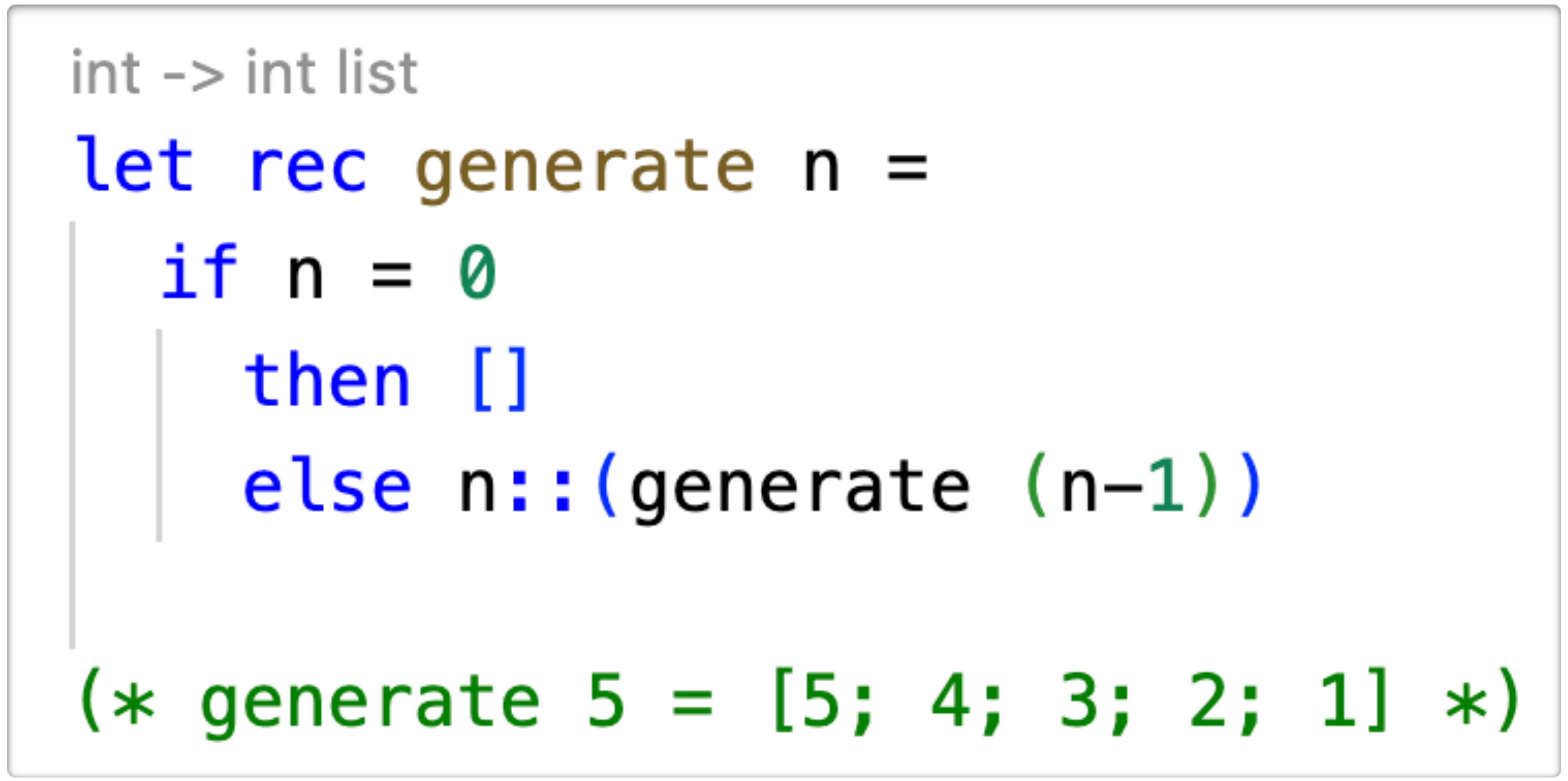
:: is right associative

So, this would be equivalent to

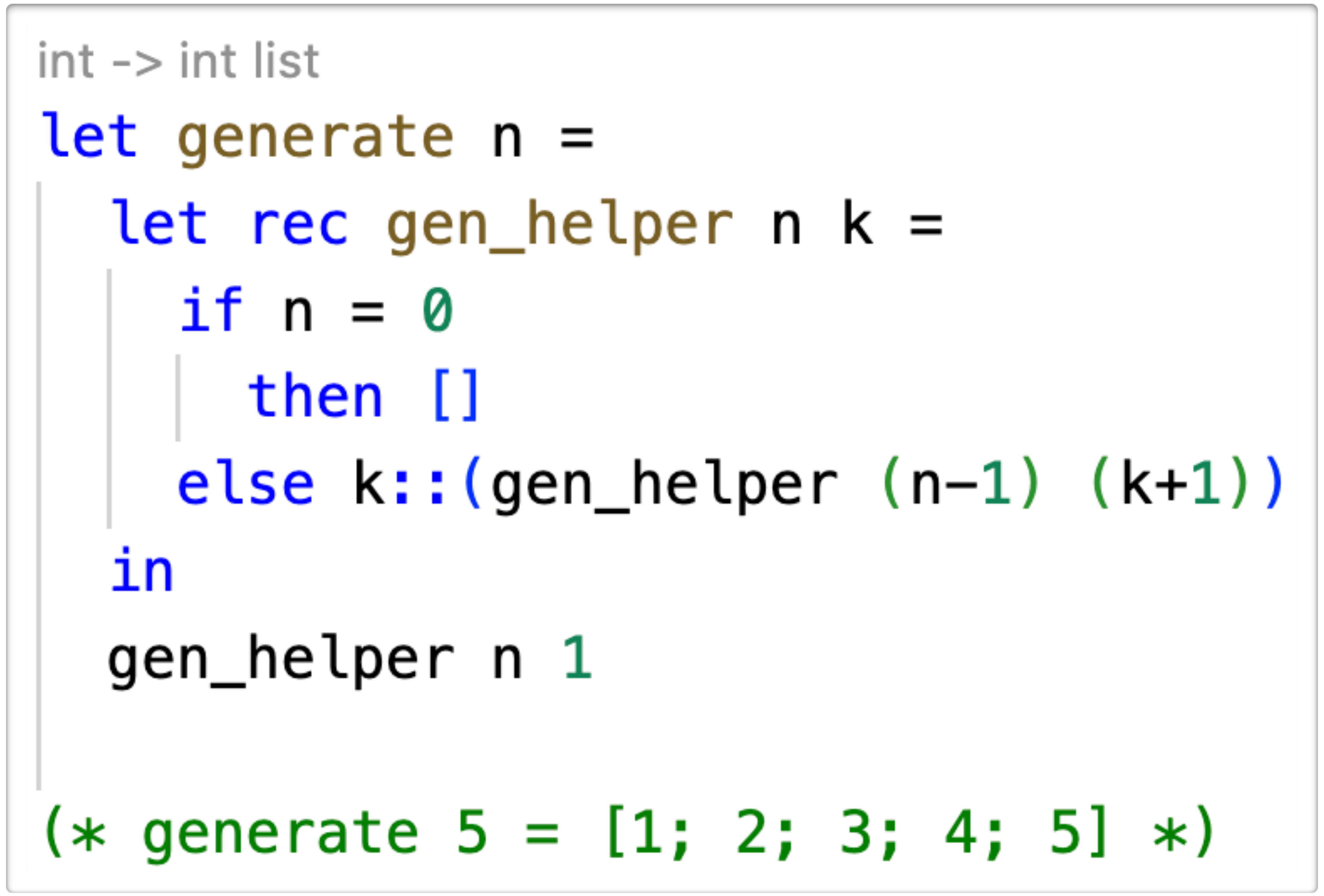
```
2::(3::l2)
```

Generating a List

» Generate a list of n natural numbers in increasing/decreasing order



```
int -> int list
let rec generate n =
  if n = 0
    then []
    else n::(generate (n-1))

(* generate 5 = [5; 4; 3; 2; 1] *)
```



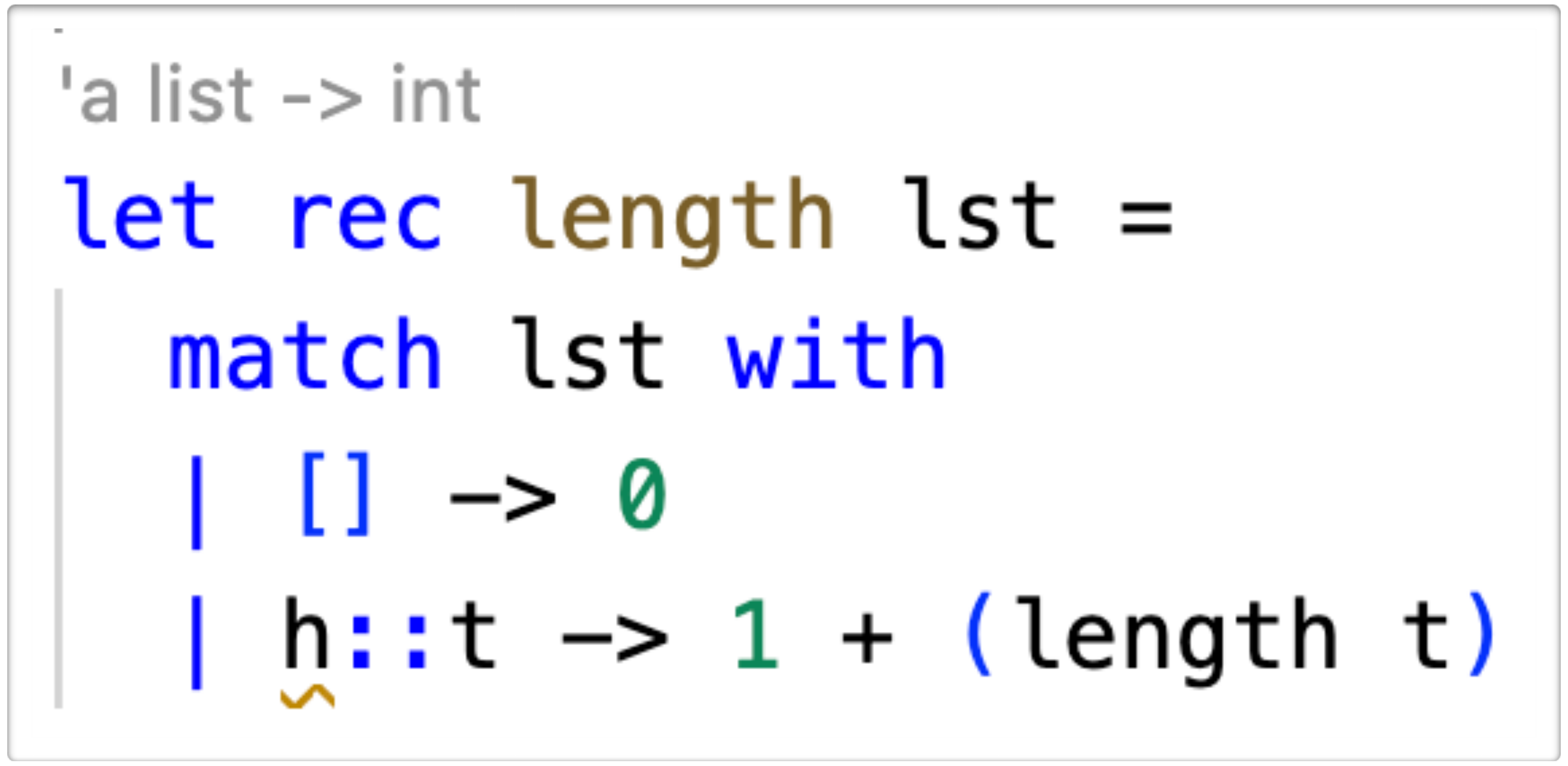
```
int -> int list
let generate n =
  let rec gen_helper n k =
    if n = 0
      then []
    else k::(gen_helper (n-1) (k+1))
  in
  gen_helper n 1

(* generate 5 = [1; 2; 3; 4; 5] *)
```

Using Lists

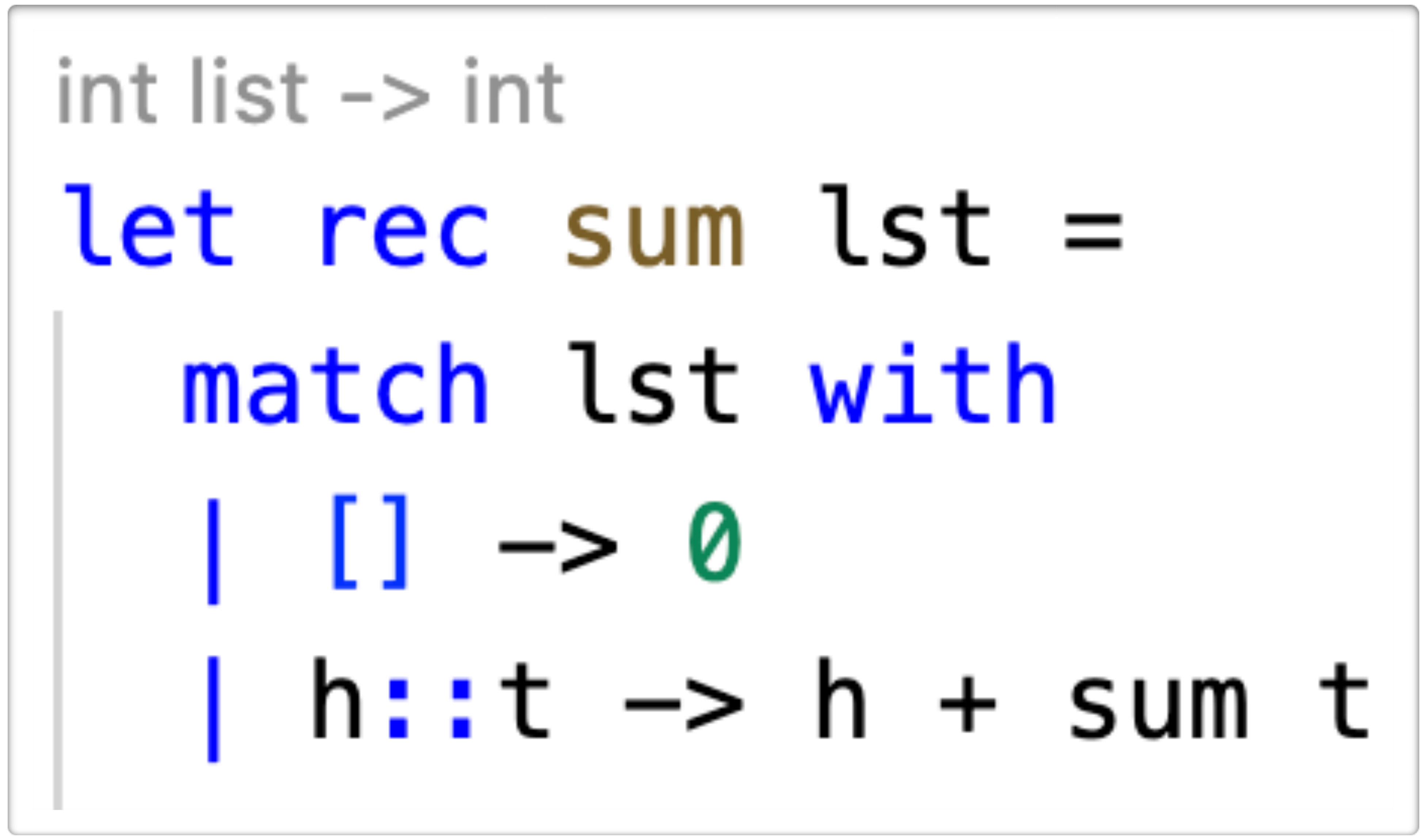
- We will use pattern–match (like a switch statement but MUCH BETTER)

```
match <list-expr> with
| [] -> <body-for-null-expr>
| h::t -> <body-for-non-null-expr>
```

Examples



```
'a list -> int
let rec length lst =
  match lst with
  | [] -> 0
  | h::t -> 1 + (length t)
```

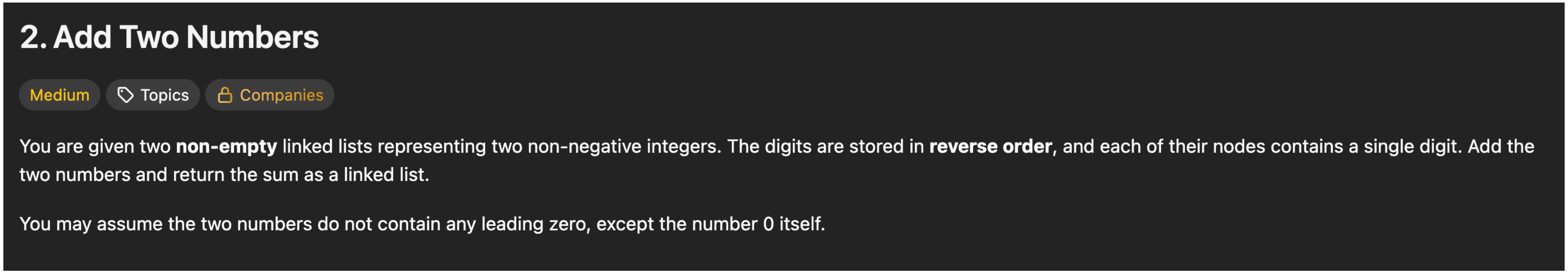


```
int list -> int
let rec sum lst =
  match lst with
  | [] -> 0
  | h::t -> h + sum t
```

demo

(basic lists)

Leetcode Medium



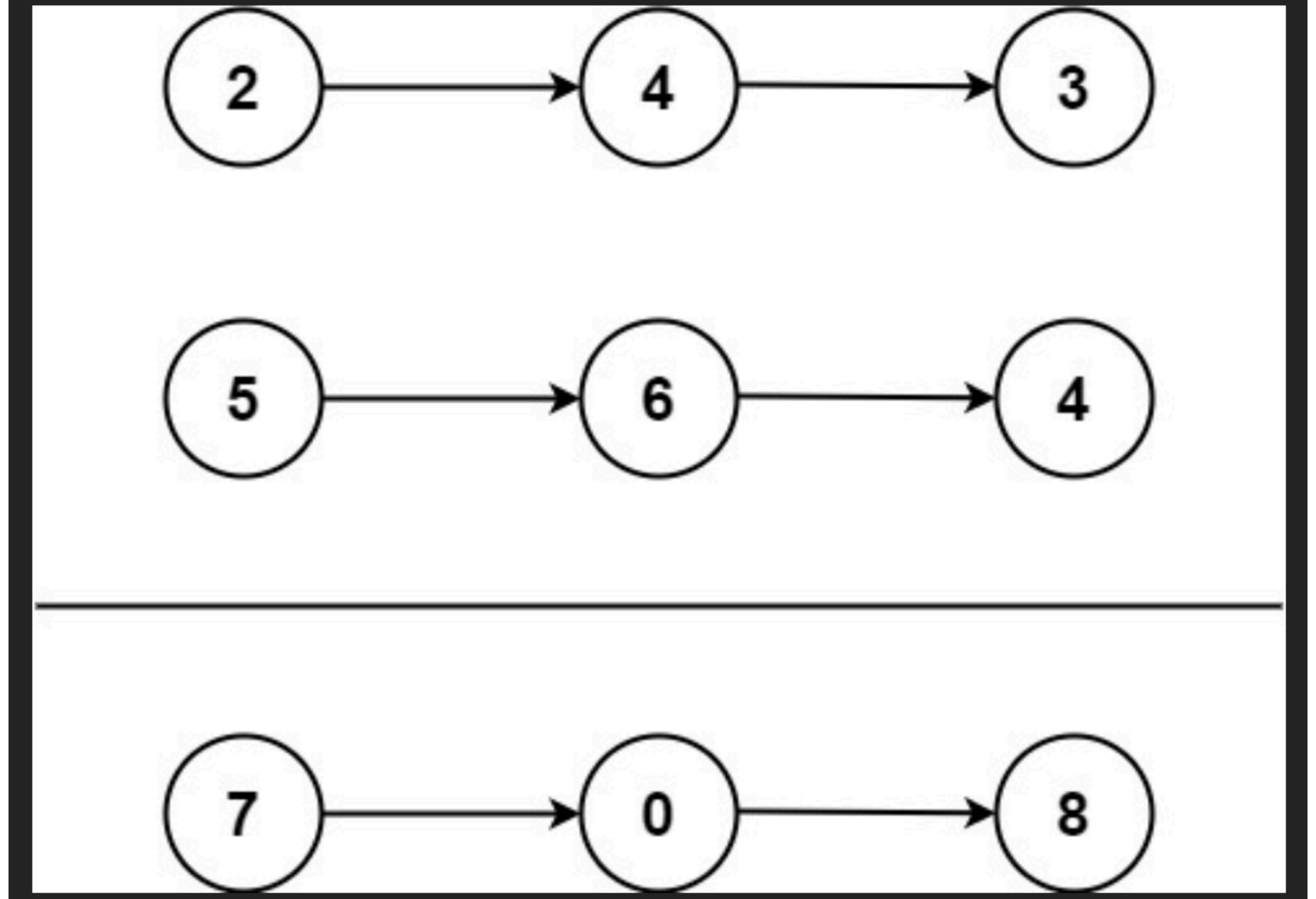
2. Add Two Numbers

Medium · Topics · Companies

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:

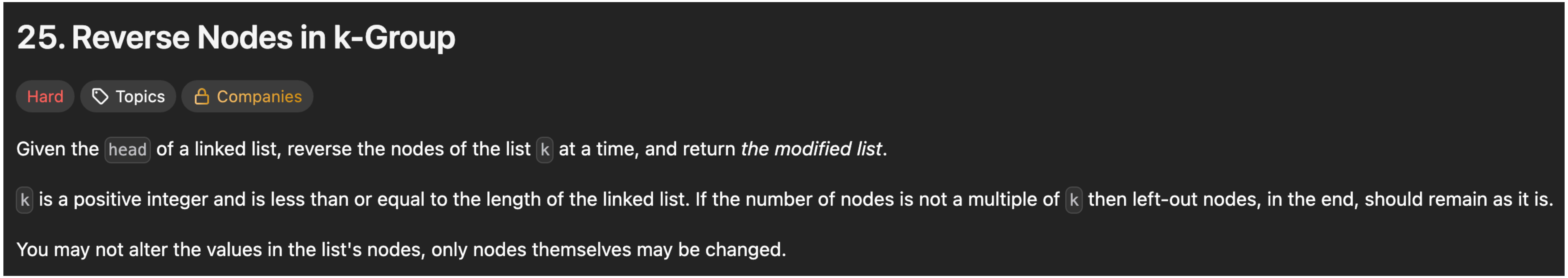


$$\begin{tikzcd} 2 \arrow[r] & 4 \arrow[r] & 3 \\ 5 \arrow[r] & 6 \arrow[r] & 4 \\ 7 \arrow[r] & 0 \arrow[r] & 8 \end{tikzcd}$$

Solution

```
let rec sum l1 l2 carry =
  match l1, l2 with
  | [], [] -> if carry = 0 then [] else [1]
  | h1::t1, h2::t2 ->
      let s = h1 + h2 + carry in
      if s >= 10 then (s-10)::(sum t1 t2 1)
      else s::(sum t1 t2 0)
  | [], h2::t2 ->
      let s = h2 + carry in
      if s >= 10 then (s-10)::(sum [] t2 1)
      else s::(sum [] t2 0)
  | h1::t1, [] ->
      let s = h1 + carry in
      if s >= 10 then (s-10)::(sum t1 [] 1)
      else s::(sum t1 [] 0);;
```

Leetcode Hard



25. Reverse Nodes in k-Group

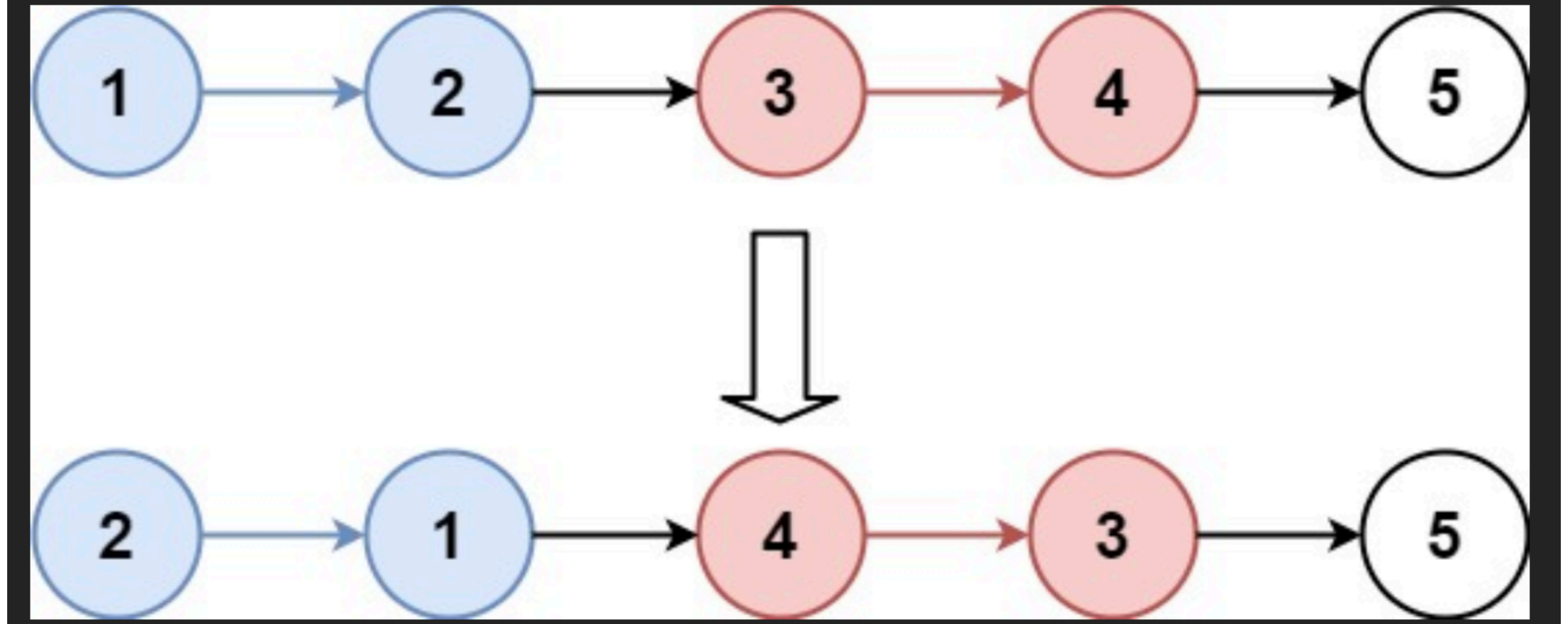
Hard · Topics · Companies

Given the `head` of a linked list, reverse the nodes of the list `k` at a time, and return *the modified list*.

`k` is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of `k` then left-out nodes, in the end, should remain as it is.

You may not alter the values in the list's nodes, only nodes themselves may be changed.

Example 1:



Solution

```
let reverse_at_k l k =
  let rec rev_k_helper l subl res k k_init =
    if k = k_init
      then rev_k_helper l [] (res @ subl) 0 k_init
    else
      match l with
      | [] -> res @ subl
      | x::t -> rev_k_helper t (x::subl) res (k+1) k_init
  in
  rev_k_helper l [] [] 0 k
```