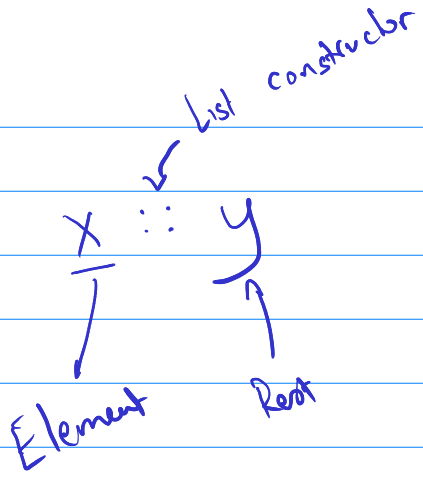


(::) : 'a → 'a list → 'a list

Not really (but this is only untrue for stupid reasons)

x @ y

(@) : 'a list → 'a list → 'a list

List List

rev (hd::tl) = (rev tl) @ [hd]

(3, 4) 3*(4+5)

(((3+4)+5) * (8+(16*9)))

Higher order functions / First class functions

Biggest Idea of FP

Curried: f x y

Uncurried: f(x y)

f x ← Partial application is completely natural when using Curried forms.

Second

add 3 4

f(x y)

add 3

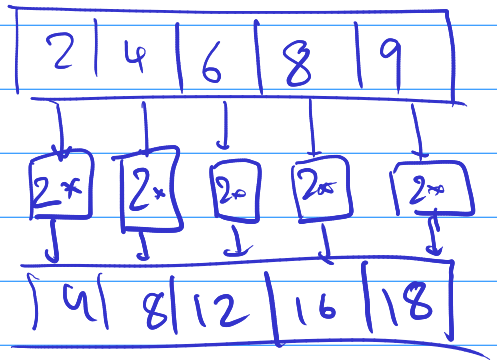
f(x ——)

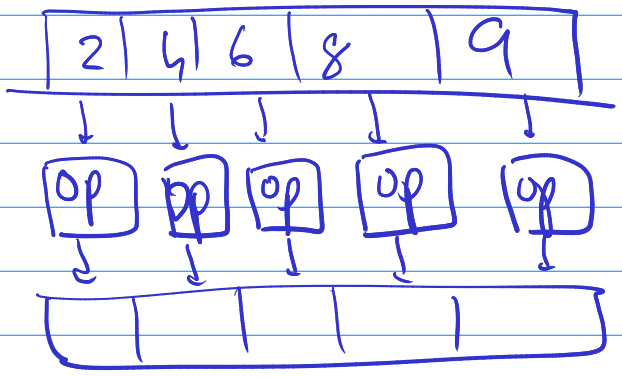
Second Big Idea of FP

- Maps & folds

1) Map

```
ans = []
for x in l:
    ans.add(2 * x)
return ans
```





```
let rec map op l =
  match l with
  | [] -> []
  | hd :: tl -> op hd :: map op tl
```

'a

map : ('a → 'b) → 'a list → 'b list

op l output

```
ans = 0
for x in l:
    ans = ans + x
return ans
```

map

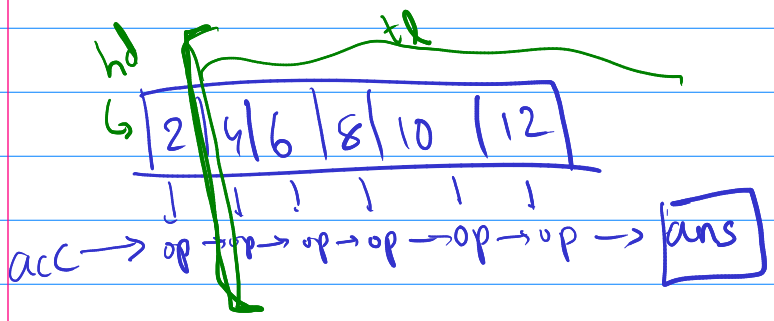
```
ans = []
for x in l:
    ans.add(op(x))
return ans
```

```
ans = 0
for x in l:
    ans = max(ans, x)
return ans
```

```
acc = some mid val
for x in l:
    acc = op(acc x)
return acc.
```

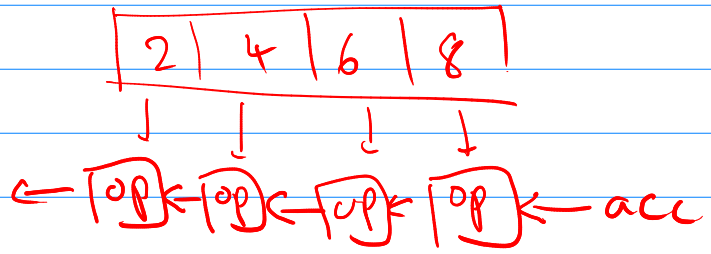
```
ans = 1
for x in l.
    ans = ans * x
return ans
```

fold l foldLR

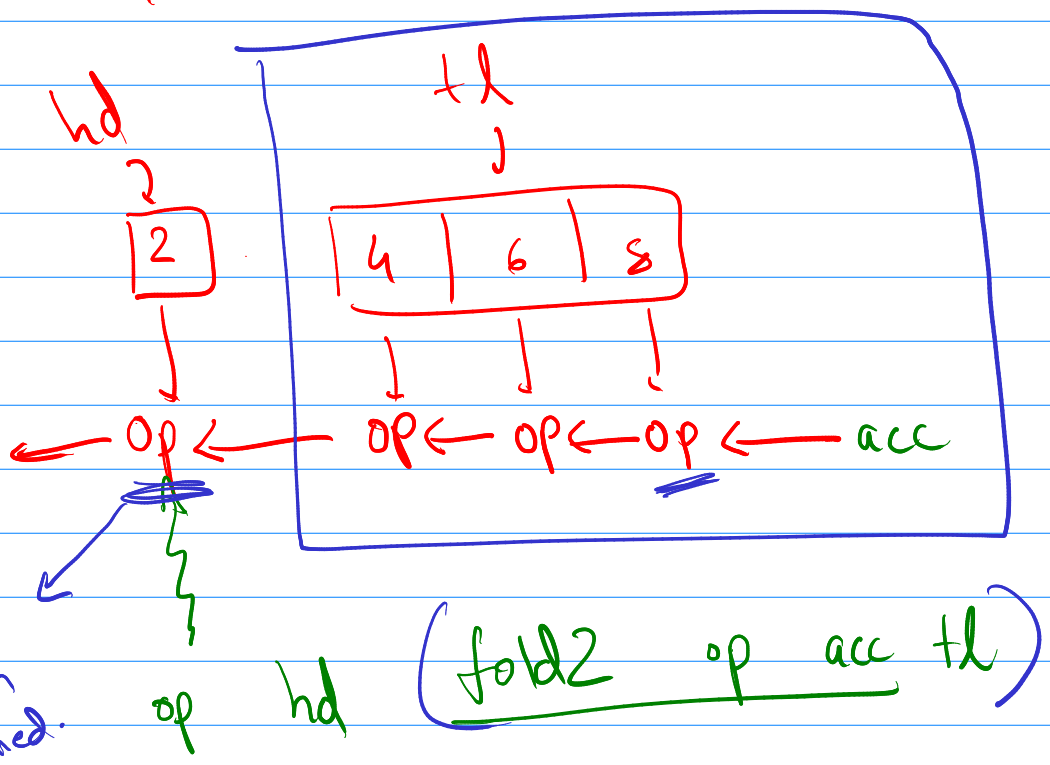


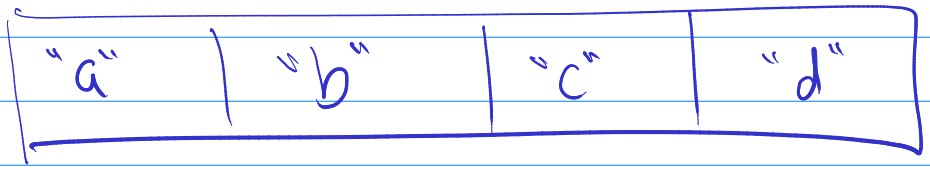

```
let rec foldl op acc l =
  match l with
  | [] -> acc
  | hd::tl -> let acc' = op acc hd in
              foldl op acc' tl
```

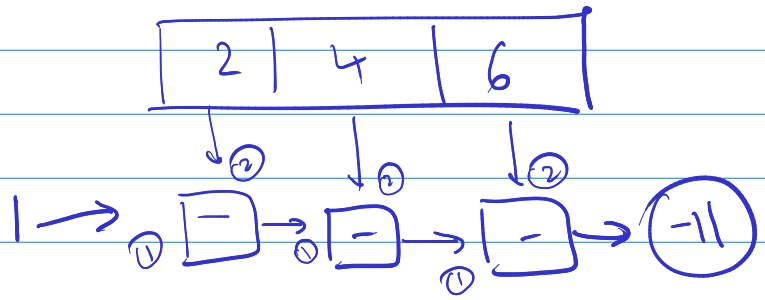
foldl op (op acc hd) tl

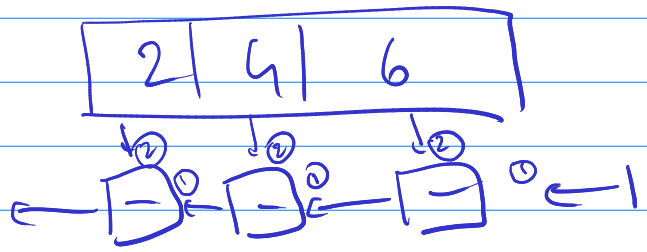


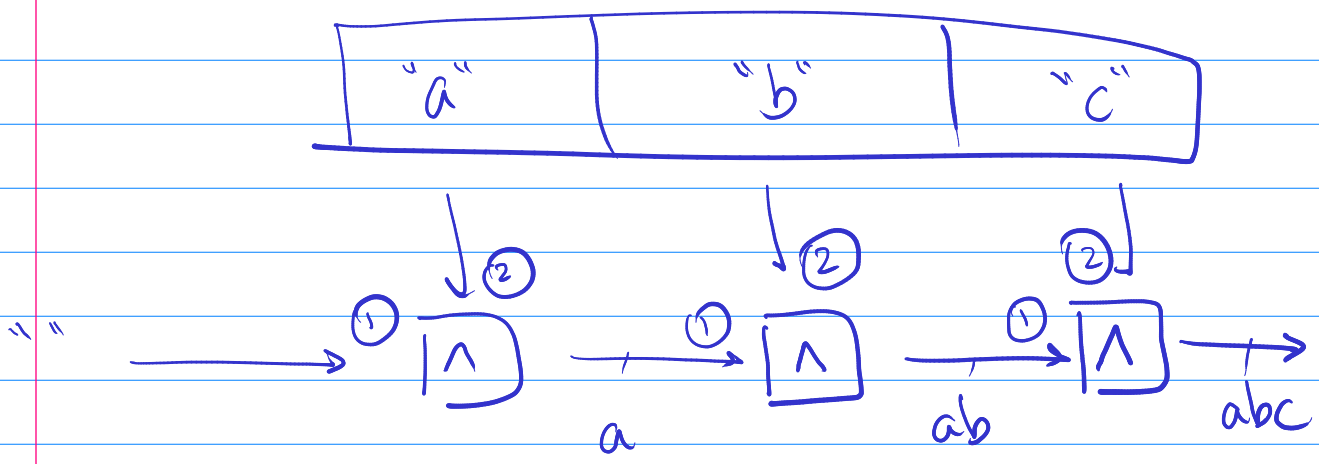

```
let rec fold2 op l acc =
  match l with
  | [] -> acc
  | hd::tl ->
```

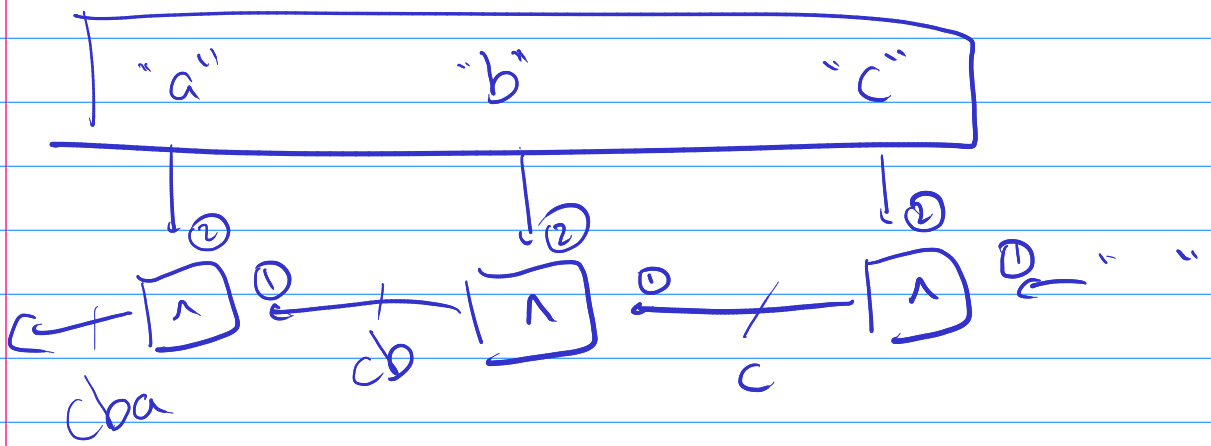


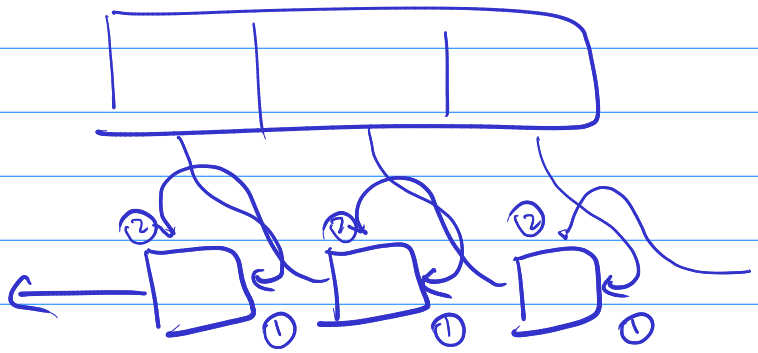

"a"
"b"
"c"
"d"


2
4
6
-11


2
4
6

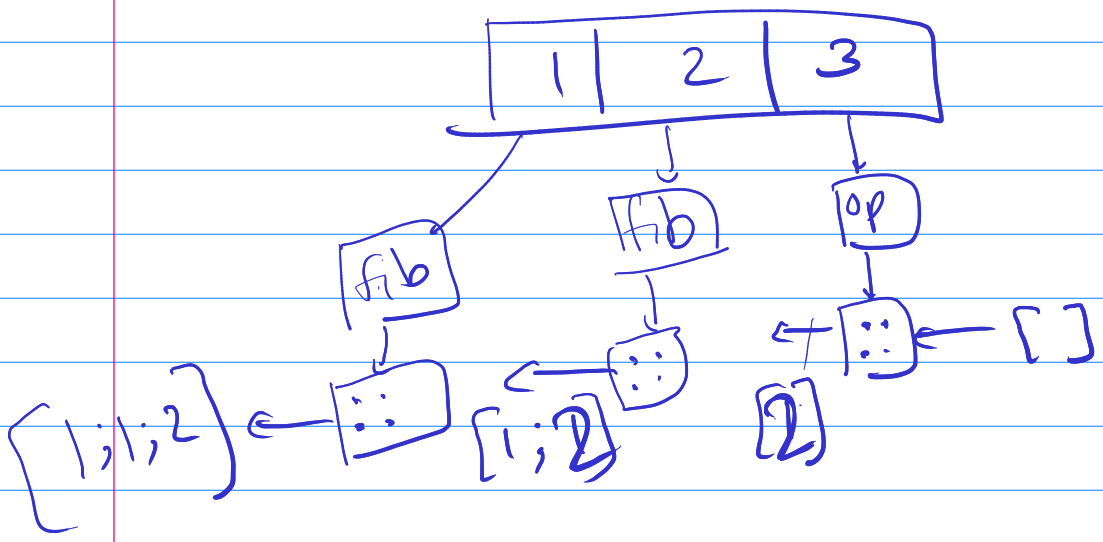

"a"
"b"
"c"
""
a
ab
abc

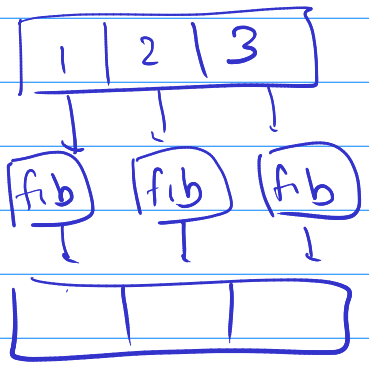

"a"
"b"
"c"
cba
cb
c
""

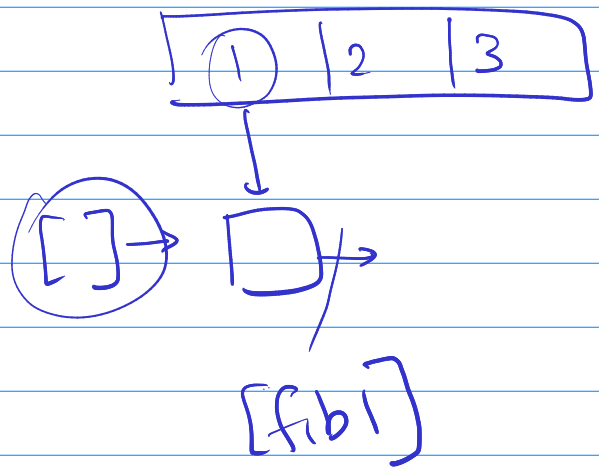


```
let mymap op l =
  List.fold_right (fun x acc -> op x :: acc) l []
```

mymap fib [1 2 3]




1
2
3
fib
fib
fib


1
2
3
[]
[fib]