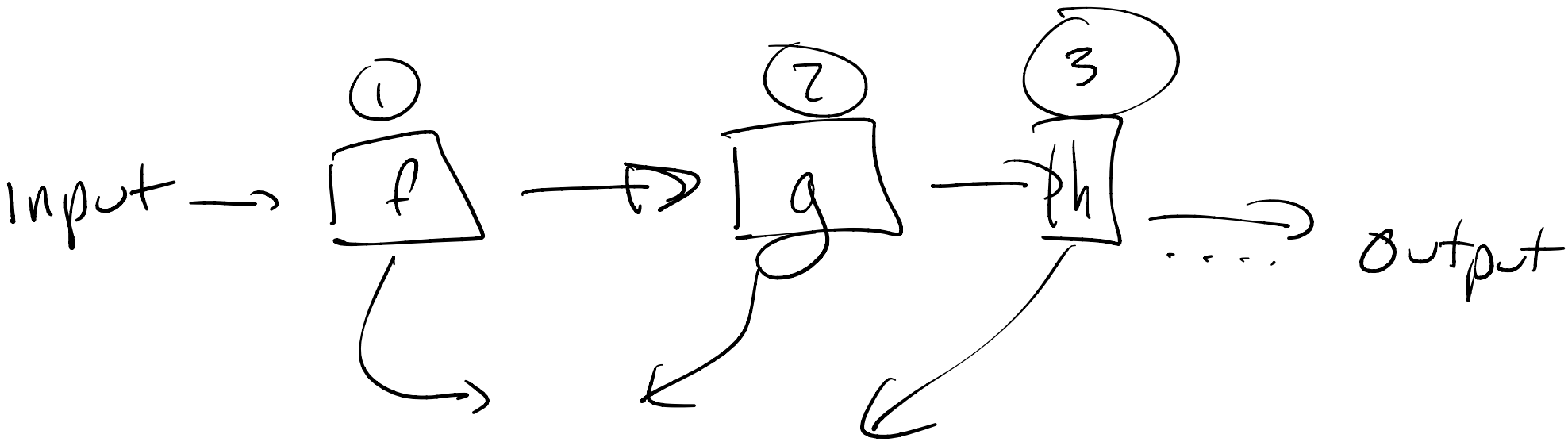
Lecture 15:

Functional Decomposition of Problems



- recursion

- uses of higher-order functions

```
fun map (f: 'a -> 'b, l: 'a list): 'b list =
    case l of
        [] => []
      | x::xs => f(x) :: map(f, xs)
```

$$map(f, [x_1 \ldots x_n]) = [f(x_1), f(x_2) \ldots f(x_n)]$$

```
fun evens(l : int list) : int list =
    case l of
        [] => []
      | x::xs => case even(x) of
                   true => x::evens(xs)
                 | false => evens(xs)
```

evens([1, 7, 6, 4, 5, 2]) = [6, 4, 2]

```
fun uppers (l : char list) : char list =
    case l of
        [] => []
      | x :: xs => case isUpper(x) of
                     true => x :: uppers(xs)
                   | false => uppers(xs)
```

uppers [#"A", #"a"] = [#"A"]

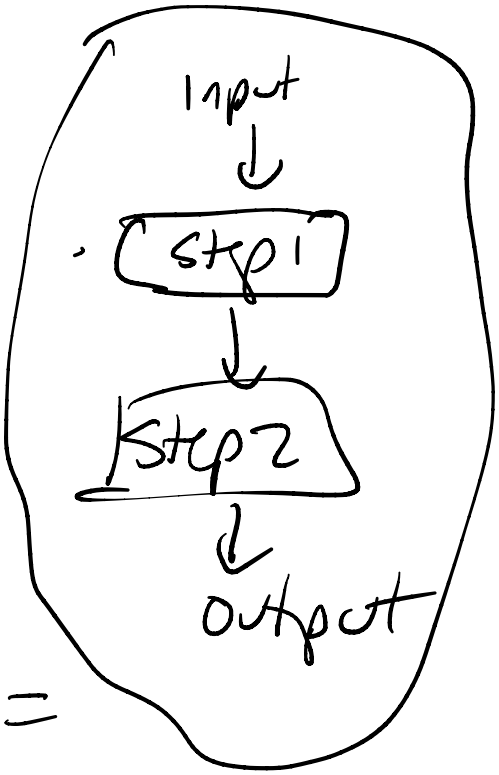
```
fun filter (p: 'a -> bool, l : 'a list) : 'a list
    case l of
      [] => []
    | x::xs => case p(x) of
                 true  => x::filter(p,xs)
               | false => filter(p,xs)
```

filter (p, l) = [x_i where p(x_i) is true]
(in the same order)

E.g. add1Lt7([1,8,7,5]) = [2,6]

```
fun add1Lt7(l: int list): int list =
    case l of
      [] => []
    | x::xs => case x<7 of
                 true => (x+1) :: add1Lt7(xs)
               | false => add1Lt7(xs)
```

```
fun addILt7(l: int list): int list =
    case l of
      [] => []
    | x::xs => case x<7 of
                 true => (x+1) :: addILt7(xs)
               | false => addILt7(xs)
```




```
fun addILt7(l: int list): int list =
  let val step1 = filter(fn x => x<7, l)
      val step2 = map(fn x => x+1, step1)
  in
    step2
  end
```

[1,8,7,5]
↓ step1
[1,5]
↓ → step2
[2,6]

```
fun addILt7(l:int list):int list =
  let val step1 = filter(fn x => x<7, l)
      val step2 = map (fn x=>x+1, step1)
  in
      step2
  end
```

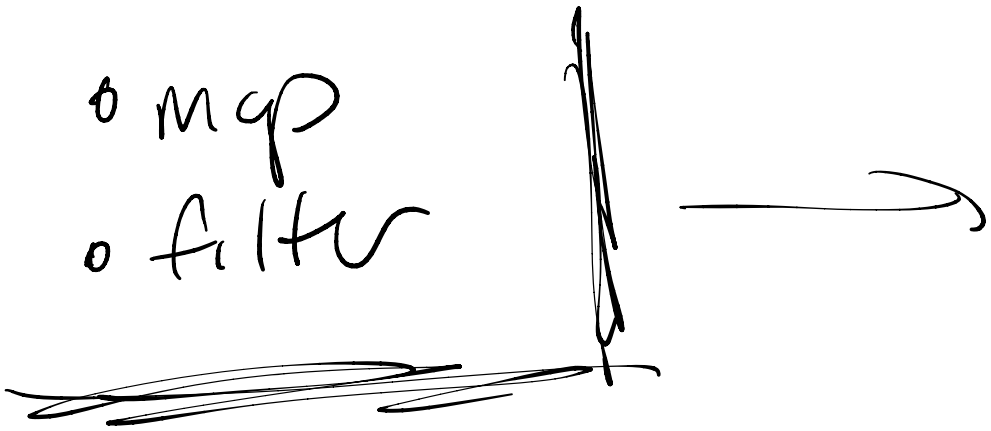
```
fun addILt7( l ) =
  map( fn x=>x+1,
       filter (fn x => x<7, l))
```

```
fun addLt7(l:int list):int list =
  let val step1 = map(fn x => x+1, l)
      val step2 = filter(fn x => x<8, step1)
  in
      step2
  end
```

[1, 8, 7, 5] —map→ [2, 9, 8, 6] —filter→ [2, 6]

can be many ways to organize the pipeline

- map
- filter



turn lists into lists

Summarize data as a value

(squish)

↳ "reduce"

```
fun sum(l: int list): int =
    case l of
        []    => 0
      | x::xs => x + sum(xs)
```

'a 'a

int

: int * int -> int

- value for []: 0
- way to combine: +

```
fun join(l: string list): string =
    case l of
        []    => ""
      | x::xs => x ^ join(xs)
```

'a 'a

string

: string * string -> string

- value for []: ""
- way to combine: ^

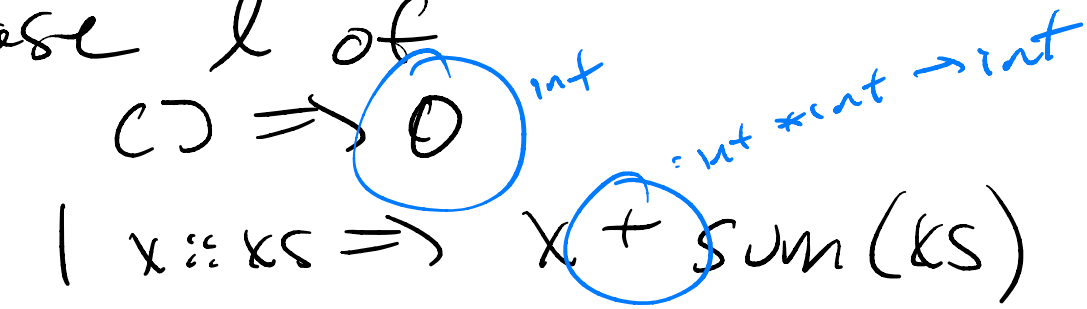
- value for empty : 'a
- combine : 'a * 'a -> 'a

```
fun reduce (combine : 'a * 'a -> 'a,
            base : 'a)
           (l : 'a list) : 'a =
case l of
    [] => base
  | x::xs => combine(x, reduce(combine, base, xs))
```

stands for 0

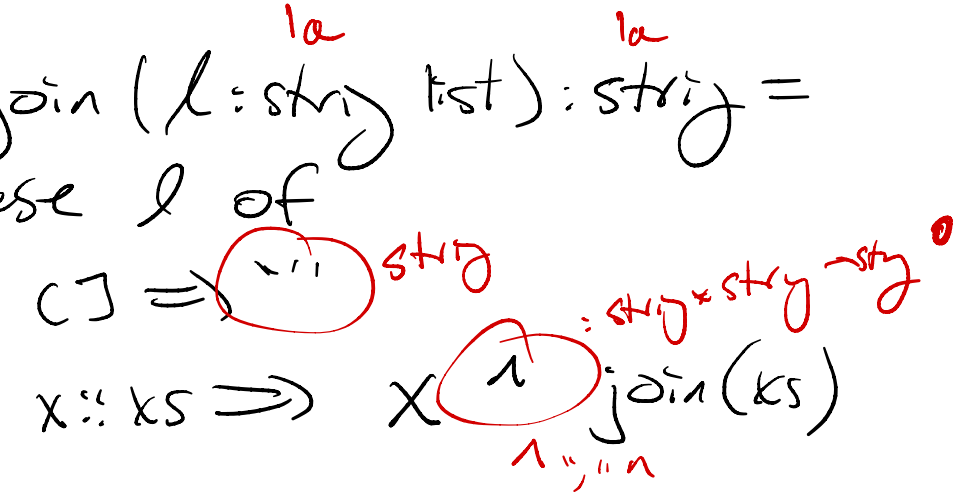
Or....

```
fun sum(l: int list): int =
    case l of
        [] => 0
      | x::xs => x + sum(xs)
```




```
fun sum l = reduce (fn (x,y) => x+y, 0, l)
```

```
fun join(l: string list): string =
    case l of
        [] => ""
      | x::xs => x ^ join(xs)
```




```
fun join(l) = reduce(fn(x,y) => x^y, "", l)
```

Find biggest number in a list

maxlist [1, 9, 21, 7] = 21 [x1, x2, x3]

x1 max(x2 max(x3 max 0))
= 0

```
fun maxlist (l:int list):int =
  reduce (
    (fn (x,y) => case x<y of
                   true => y
                 | false => x),
    minInt (the smallest negative integer) -∞
    l)
```

max

Stock Price

buy sell buy sell

$40, $20, $0, $1, $3, $9, $21

day1 day2 day3 day4 day5 day6 day7] input

best gain (in retrospect)

-20

+21

$40, $20, $0, $1, $3, $9, $21

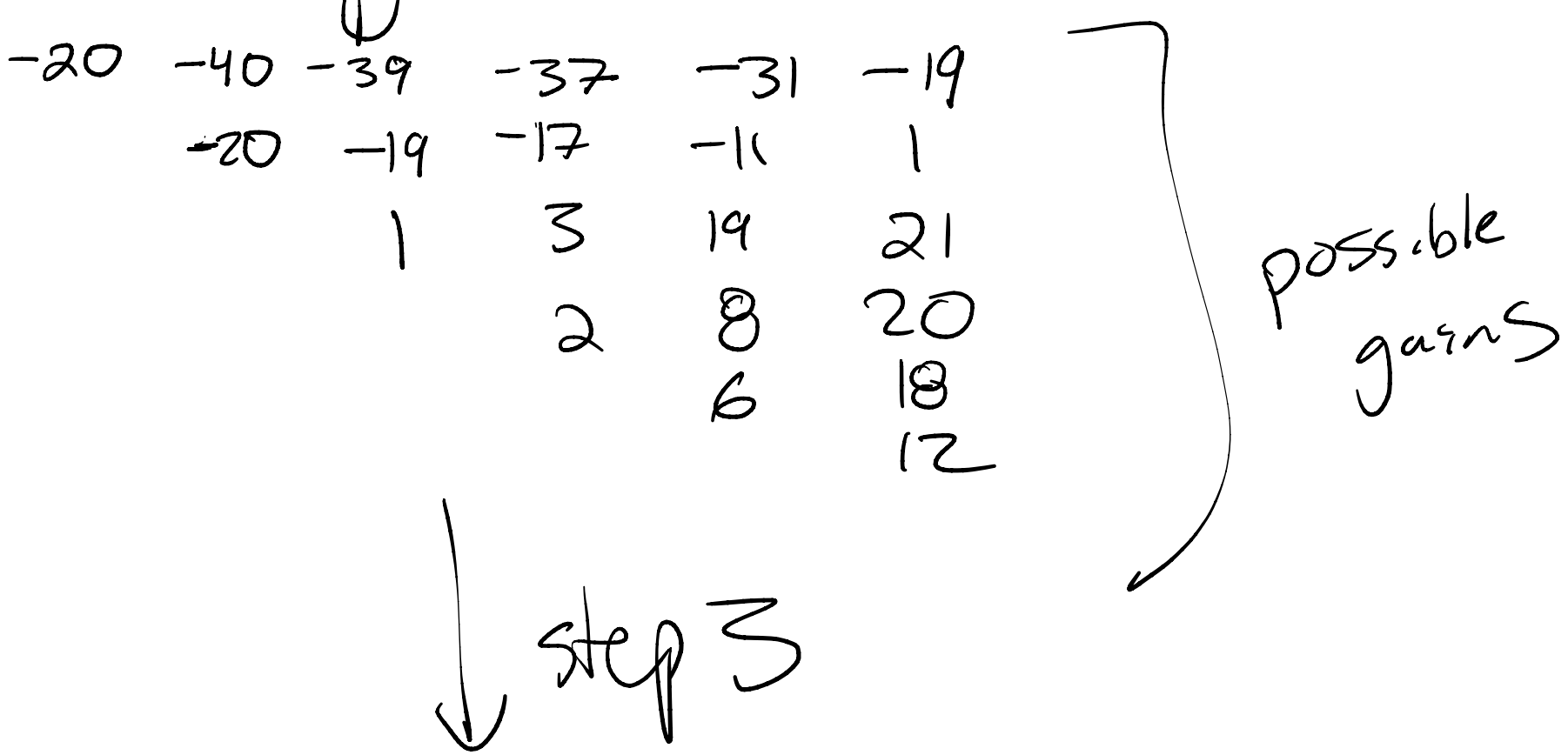
Step 1

$40	20 0 1 3 9 21
$20	0 1 3 9 21
0	1 3 9 21
1	3 9 21
3	9 21
9	21
21	

$40	20 0 1 3 9 21
$20	0 1 3 9 21
0	1 3 9 21
1	3 9 21
3	9 21
9	21
21	

Step 2

-20	-40	-39	-37	-31	-19
	-20	-19	-17	-11	1
		1	3	19	21
			2	8	20
				6	18
					12


0
-20 -40 -39 -37 -31 -19
-20 -19 -17 -11 1
1 3 19 21
2 8 20
6 18
12
possible gains

step 3

21

$40, $20, $0, $1, $3, $9, $21

↓ Step 1

$40	20 0 1 3 9 21
$20	0 1 3 9 21
0	1 3 9 21
1	3 9 21
3	9 21
9	21
21	

[40, 20, 0, 1, 3, 9, 21]

x xs

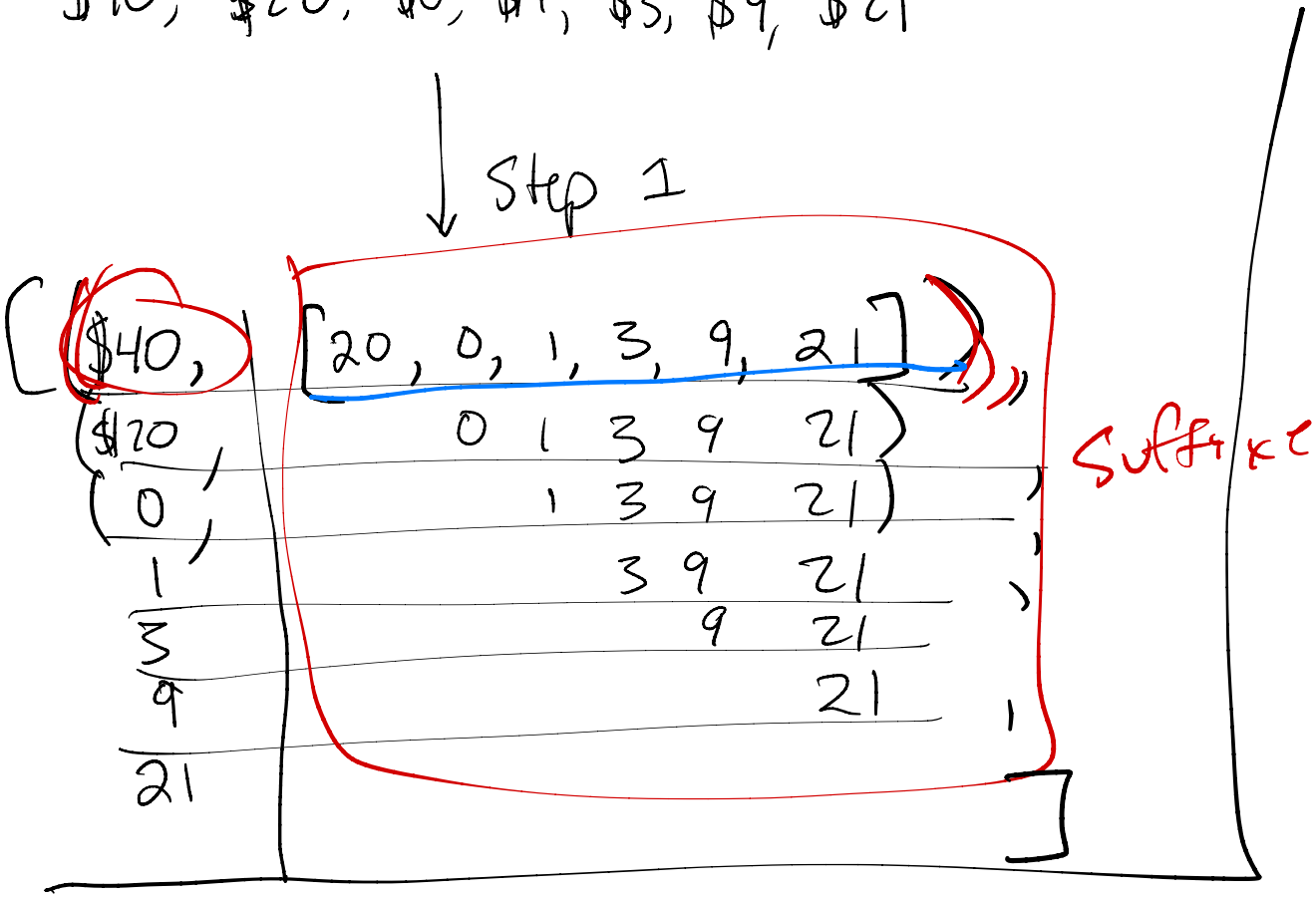
[[20, 0, 1, 3, 9, 21], ...]

```
fun suffixes (l : int list) : (int list) list =
  case l of
    [] => []
  | x::xs => xs :: suffixes(xs)
```

xs : int list

suffixes(xs) : (int list) list

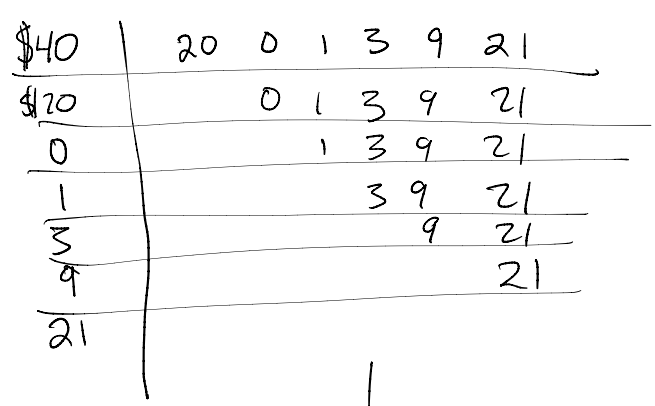
$40, $20, $0, $1, $3, $9, $21



zip : ('a list * 'b list)
→ ('a * 'b) list

row

```
fun withSuffixes (l : int list) : (int * (int list)) list =
    zip (l, suffixes l)
```



buy = $40

sells = [20, 0, 1, 3, 9, 21]

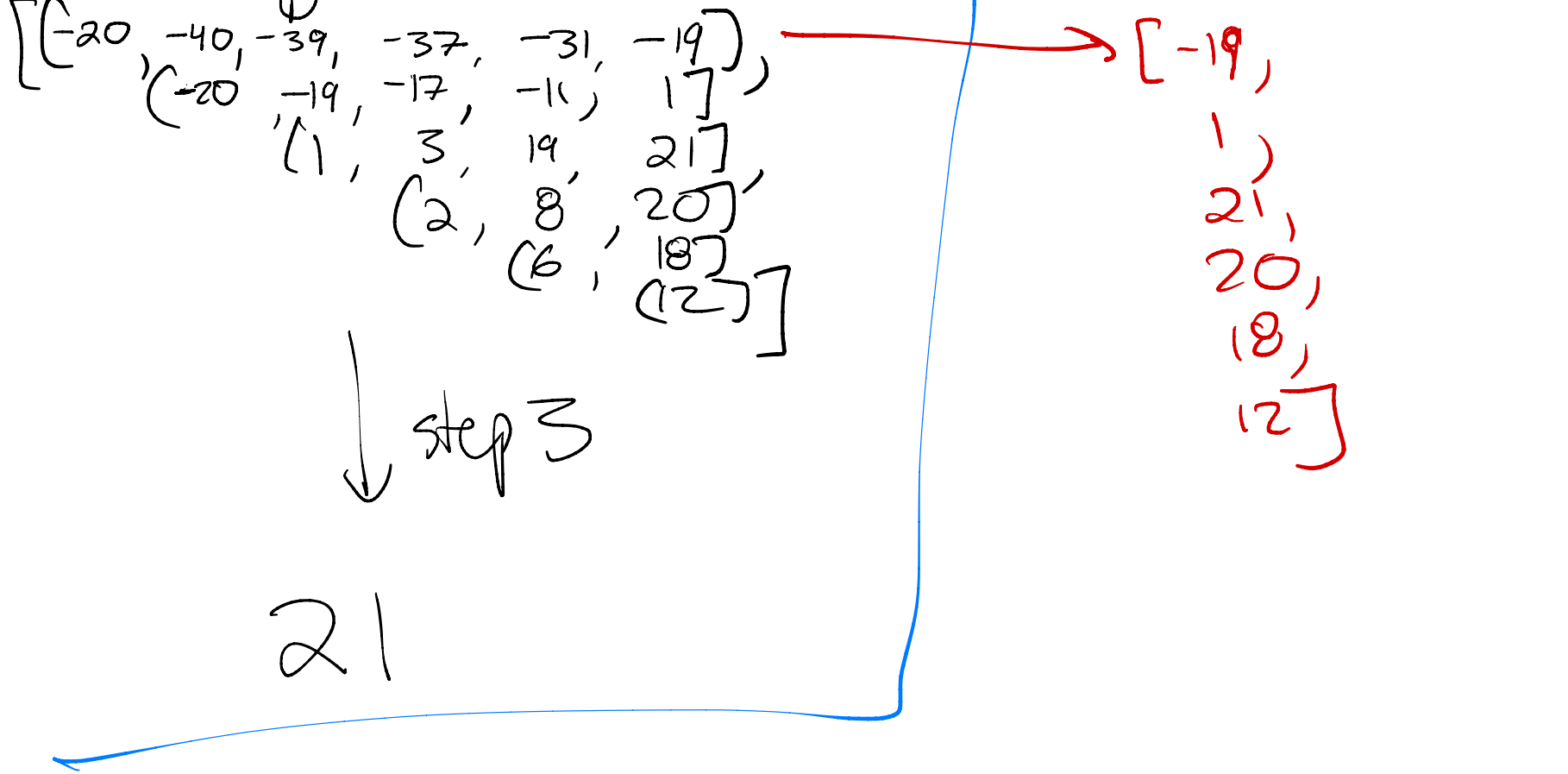
Step 2

[(-20, -40, -39, -37, -31, -19),
(-20, -19, -17, -11, 1),
(1, 3, 19, 21),
(2, 8, 20),
(6, 18),
(12)]

```
fun gains(buy-sells: (int * (int list)) list): (int list) list =
  map(fn (buy, sells) =>
        map(fn sell => sell - buy, sells),
      buy-sells)
```

int list

"closure"




```
fun maxAll(l:(int list) list):int =
    maxlist(map(maxlist, l))
```

```
fun bestGain(l: int list): Int =
    maxAll(gains(withSuffixes l))
```