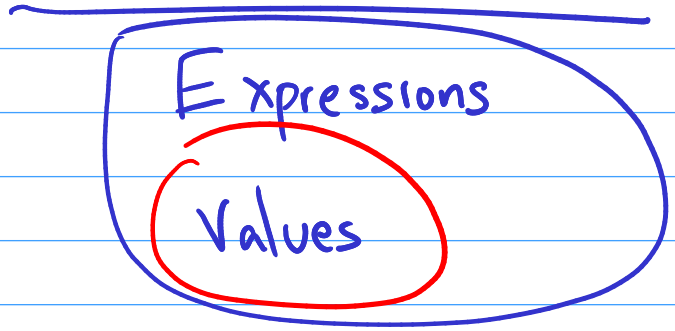
Types



P1: You're given a list

[1 2 3 4 5]

Find all even numbers in this list

```
let rec allEven l =
  match l with
  | [] -> []
  | hd :: tl -> if hd mod 2 = 0 then hd :: allEven tl else allEven tl
```

P2: A number is big if it is greater than 50.

Find all big numbers in the list.

```
let rec allBig l =
  match l with
  | [] -> []
  | hd :: tl -> if hd > 50 then hd :: allBig tl else allBig tl
```

P3: Find all prime numbers in a list.

```
let rec allPrimes l =
 match l with
 | [] -> []
 | hd :: tl -> if isPrime hd then hd :: allPrimes tl else allPrimes tl
```

```
let isPrime n =
  let rec helper cnt = if cnt = 1 then true else if n mod cnt = 0 then false else helper (cnt - 1) in
  helper (n - 1)
```

Can we write an abstract fn which does this filtering for us?

```
let rec filter f l =
  match l with
  | [] -> []
  | hd :: tl -> if f hd then hd :: filter f tl else filter f tl
```

f : 'a -> bool

can be any predicate

"First-class functions"

P4 Double each number in a list

```
let rec doubleAll l =
 match l with
 | [] -> []
 | hd :: tl -> (2 * hd) :: doubleAll tl
```

P5 Download all webpages from a list

```
let rec downloadAll l =
 match l with
 | [] -> []
 | hd :: tl -> let u = download hd in u :: downloadAll tl
```

```
let rec downloadAll l =
 match l with
 | [] -> []
 | hd :: tl -> (download hd) :: downloadAll tl
```

```
let download url = print_endline ("Downloaded " ^ url)
```

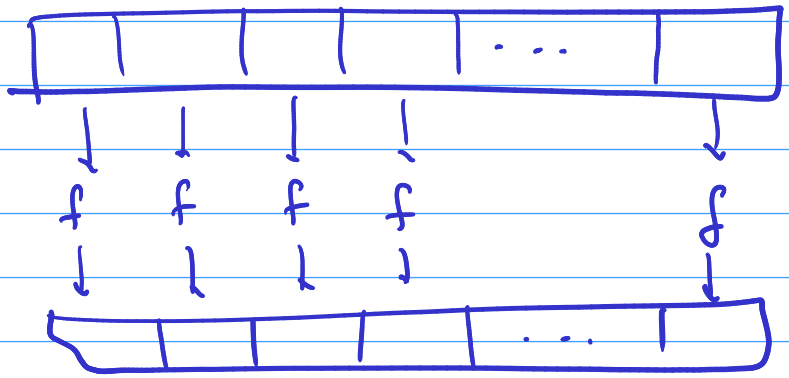
Do something to each element of a list

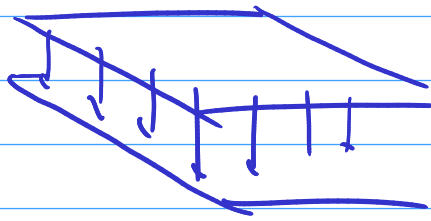
```
let rec map f l =
 match l with
 | [] -> []
 | hd :: tl -> (f hd) :: map f tl
```

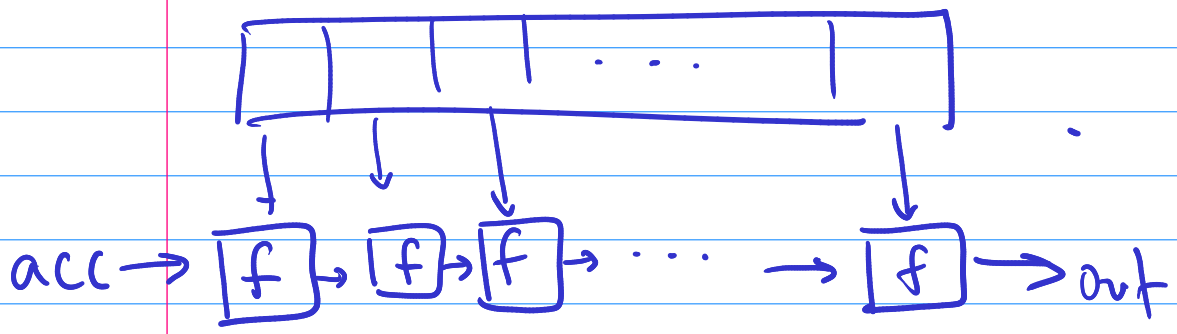
```
9 l = [1, 2, 3, 4, 5]
10 ans = []
11 for x in l:
12     ans.append(f(x))
```

```
let doubleAll l = map double l
```

```
let downloadAll l = map download l
```







reduce
fold