Contracts for Functional Arguments and Polymorphism
Highlights from last lecture

- One function:

```
;; below : Ion number  ->  Ion
;; to construct a list of those numbers
;; in alon that are below t
(define (below alon t)
  (cond [(empty? alon) empty]
        [else (cond [(< (first alon) t)
                       (cons (first alon) (below (rest alon) t))]
                 [else (below (rest alon) t)]))])
```
Another function:

;; above : lon number  ->  lon
;; to construct a list of those numbers
;; in alon that are above t
(define (above alon t)
  (cond [(empty? alon) empty]
        [else (cond [(> (first alon) t)
                                (cons (first alon) (above (rest alon) t))]
                  [else (above (rest alon) t)])]))
Highlights from last lecture

- Capture the pattern:

```lisp
;; filter : comparison, lon number -> lon
;; to construct a list of those numbers n
;; in alon such that (test t n) is true
(define (filter test alon t)
  (cond [(empty? alon) empty]
        [(test (first alon) t)
         (cons (first alon) (filter test (rest alon) t))]
        [else (filter test (rest alon) t)])
)```
Highlights from last lecture

- Write things more concisely:
- The magic moment:

  \[
  \begin{align*}
  & \text{(define} (\text{above} \ \text{alon t}) \ (\text{filter} \ > \ \text{alon t})) \\
  \text{and} & \text{(define} (\text{below} \ \text{alon t}) \ (\text{filter} \ < \ \text{alon t}))
  \end{align*}
  \]

- Both functions will work just as before
Just for the record:

- Syntactically, is this already allowed?

\[
\text{(define (above alon t) (filter > alon t))}
\]

\[
\text{(define (below alon t) (filter < alon t))}
\]
What’s “comparison”? 

;; filter : (number, number -> boolean), lon number -> lon 
;; to construct a list of those numbers n 
;; in alon such that (test t n) is true 
(define (filter test alon t) 
  (cond [(empty? alon) empty] 
    [else (cond [(test (first alon) t) 
                (cons (first alon) (filter test (rest alon) t))]
                [else (filter test (rest alon) t)])])))
Can we do better?

;; filter : (X, Y => boolean), [X], Y => [X]
;; to construct a list of those numbers n
;; in alon such that (test t n) is true
(define (filter test alon t)
  (cond [(empty? alon) empty]
    [else (cond [(test (first alon) t)
                (cons (first alon) (filter test (rest alon) t))]
              [else (filter test (rest alon) t)])])))
Examples:

- append: [X], [X] -> [X]
- length: [X] -> natural
- my-first: [X] -> X
- my-rest: [X] -> [X] or false
- my-cons: X, [X] -> [X]
- my-empty: [X]
- compose-functions: ?
We’re going real fast: do the reading!
Lab is very important this week
Homework will be available later today
  To take into account how last one worked
Quiz will cover last two lectures
  Will be available later today
  A bit longer than usual
  Due by end of day tomorrow