



More on Lists



Today's Lecture

- We've seen
 - `length` : `[any]` -> `natural`
 - `add`, `mult`, `count-u`, `sum-of-squares`, `average` : `[num]` -> `num`
- Today:
 - `map-double`, `map-square`: `[num]` -> `[num]`
 - `append` : `[X]`, `[X]` -> `[X]`
 - `flatten` : `[[X]]` -> `[X]`



Natural Template for lists

- Templates should correspond to contracts:

```
(define (f x)
```

```
(cond
```

```
[(empty? x) ...]
```

```
[else (...(first x)...(f (rest x))...)]))
```



Returning lists

```
; map-double : [num] -> [num]
```

```
; Example:
```

```
; (map-double (cons 3 (cons 4 empty)))
```

```
; → (cons 6 (cons 8 empty))
```

```
(define (map-double x)
```

```
  (cond
```

```
    [(empty? x) empty]
```

```
    [else (cons (* 2 (first x))
```

```
              (map-double (rest x)))]))
```



Similarly

```
; map-square : [num] -> [num]
```

```
; Example :
```

```
;      (map-square (cons 3 (cons 4 empty)))  
;      → (cons 9 (cons 16 empty))
```

```
(define (map-square x)  
  (cond  
    [(empty? x) empty]  
    [else (cons (sqr (first x))  
                (map-square (rest x)))]))
```



Append

; append : [X], [X] -> [X]

; Example : (append [1,2,3] [4,5,6])

; → [1,2,3,4,5,6]

```
(define (append x y)
```

```
  (cond
```

```
    [(empty? x) y]
```

```
    [else (cons (first x)
```

```
            (append (rest x) y))]))
```