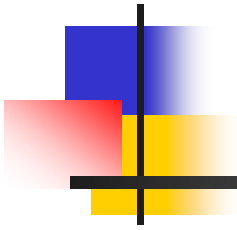


Functional Arguments and Polymorphism at Work





Look for the pattern

- One function:

```
; my-fun1 : [number] -> [number]
```

```
; adds one to each number in list
```

```
(define (my-fun1 l)
```

```
  (cond [(empty? l) empty]
```

```
        [else (cons (add1 (first l))
```

```
                    (my-fun1 (rest l)))]))
```



Look for the pattern

- Another function function:

```
; my-fun2 : [boolean] -> [boolean]
```

```
; inverts each boolean in the list
```

```
(define (my-fun2 l)
```

```
  (cond [(empty? l) empty]
```

```
        [else (cons (not (first l))
```

```
                    (my-fun2 (rest l)))]))
```



Codify the pattern

- Another function function:

; `map` : $(X \rightarrow Y), [X] \rightarrow [Y]$

; applies `f` to each element in `l`

```
(define (map f l)
```

```
  (cond [(empty? l) empty]
```

```
        [else (cons (f (first l))
```

```
                    (map f (rest l))))])
```



Use the pattern

- `(define (f-1 l) (map add1 l))`
- `(define (f-2 l) (map not l))`
- `(define (f-3 l) (map sqr l))`
- `(define (f-4 l) (map length l))`
- `(define (f-5 l) (map first l))`
- `(define (f-6 l) (map symbol? l))`
- `(define (f-7 l) (map swap l))`



Templates as functions

- Recall the template for lists:

```
; (define (fun-for-l l)
;   (cond [(empty? l) ...]
;         [else ... (first l)
;                   ... (fun-for-l (rest l)) ...]))
```

- Can we pass the "... "s as parameters?



Templates as functions

- It would look just like this:

```
(define (fun-for-l p1 p2 l)
  (cond [(empty? l) p1]
        [else (p2 (first l)
                    (fun-for-l p1 p2 (rest l)))]))
```

- What does this mean for all the functions we've written over lists?



Templates as functions

- It would look just like this:

```
(define (fun-for-l p1 p2 l)
  (cond [(empty? l) p1]
        [else (p2 (first l)
                    (fun-for-l p1 p2 (rest l)))]))
```

- What's a good type for this gadget?
 - $X (Y X \rightarrow X) [Y] \rightarrow X$