Functional Arguments/Values

→ `(DEFUN applier (fn) (fn 5 11))`

`applier`

→ `(applier '+)`

16

→ `(applier '*)`

55

→ `(applier '(LAMBDA (x y) (+ (* 2 x)(- y 3))) )`

18

Notice:
value of `fn` is a FUNCTION;
which can be evaluated on args!

Example of Functional Args

`(LAMBDA (x y fn) (fn x (CDR y)))`

of	`A`	&	`(B C D)`	&	`CONS`	is	
of	`(t t)`	&	`(t)`	&	`CONS`	is	
of	`5`	&	`(A)`	&	`CONS`	is	
of	`(A B C)`	&	`(D E F)`	&	`APPEND`	is	
of	`(A B C)`	&	`(t)`	&	`APPEND`	is	
of	`A`	&	`(B C)`	&	`(LAMBDA (x y) x)`	is	

Sufficient?

- Any computation:
  `(f` $v_1 \cdots v_n$`)`

- No need to *name* `f` ! (Just write it!)
  `( (LAMBDA (`$a_1 \cdots a_n$`) form)` $v_1 \cdots v_n$`)`

- What if "F' calls "G"?
  Just EMBED `G`'s code within `F`!
  Given `g: z` ⇝ `(CONS (CDR z) (CAR z))`
  `f: x,y` ⇝ `(EQ (g x) (CAR y))`
  Write as
  `f: x,y` ⇝ `(EQ ((LAMBDA (z) (CONS (CDR z) (CAR z)))`
  `x)`
  `(CAR y))`

  (Works whenver f calls g calls h calls . . .)
  Note: No `SETQ`s, No `DEFUN`s

- But: What if function calls ITSELF!

Problem with Recursion

For $g: \; x \rightsquigarrow \begin{cases} 1 & \text{if } x = 0 \\ x \times \underline{g}(x-1)) & \text{if } x > 0 \end{cases}$

```
(LAMBDA (x)
     (COND  ((EQ x 0)   1)
            ( t         (TIMES x (?? (- x 1)))))
```

(Problematic even if indirect:
f calls g calls h calls $\underline{f}$)

Labels

- `(LAMBDA (x) ··· x ··· )`
  uses `x` to temporarily
  name 1^{st} arguments

- `(LABEL foo (LAMBA ··· foo ··· )`
  uses `foo` to temporarily
  name λ-expr (containing `foo`)

- Allows a function to call itself
  RECURSIVELY!

- Examples: `FIRST` p33
  `MEMQ` p38

NOTE: `Label` not in all *Lisp* dialects
not in Franz!

Example of `Label`: INIT

- Function to return INITial atom
  in S-expression:

A	$\leadsto$ A
((A B) C)	$\leadsto$ A
((((A) B) C D) E)	$\leadsto$ A

- Answer:

```
(LABEL init (LAMBDA (x)
   (COND ((ATOM x) x    )
         ( t       (init (CAR x)) ))))
```

- Kinda like:

```
(LAMBDA (x)
   ((LAMBDA (init) (init x))
    '(LAMBDA (x) (COND ((ATOM x) x   )
                       ( t       (init (CAR x)) ))))
```


Example of `Label`: MEMQ

- `memq` function of 2 args: `at` [atom], `l` [list]:
  returns { `t` if `at` is a member of `l`; `nil` otherwise }

- Code:

```
(LABEL memq (LAMBDA (at l)
  (COND  ((NULL l)           nil    )
         ((EQ (CAR l) at)    t       )
         ( t                 (memq at (CDR
  ))
```

- Examples of use of `MEMQ`:

```
→ ( (LABEL memq (LAMBDA (at l) ...))  'A '(B A C) )
t
→ ( (LABEL memq (LAMBDA (at l) ...))  'Q '(t 4) )
nil
→ ( (LABEL memq (LAMBDA (at l) ...))  'Q '(Q) )
t
```


Avoiding `Label`

Usually use *PERMANENT NAMING*

```
 → (DEF memq (LAMBDA (at l)
  (COND   ((NULL l)          nil   )
          ((EQ (CAR l) at)   t   )
          ( t                (memq at (CDR l)) )
   ) ))
memq
```

→ (memq 'A '(B A C))

t

→ (memq 'Q '(t 4))

nil

→ (memq 'Q '(Q))

t

All of Pure Lisp

- A *FUNCTION* is
  - Primitive [atomic]: `CONS`, `CAR`, `EQ`, ...
  - λ-expression: `(LAMBDA (`v_1 ... v_n`)` ⟨form⟩`)`
  - Variable (evaluating to Function)
  - Labelled λ-expression: `(LABEL atm` λ`-expr)`

- A *FORM* is
  - Atom [constant or variable]: `nil`, `5`, `X`
  - Function application: `(fn` f_1 ... f_n`)`
  - Quoted expr: `(QUOTE s)`
  - Cond expr: `(COND (`p_1 q_1`)` $\cdots$ `(`p_n q_n`) )`