Functional Programming

- Functions
- *Lisp*'s Foundations
  Data Structures
  Built-in Functions + Predicates
  Evaluation
  Lambda-Expressions
  Special Forms
  Functional Arguments + Label Lambda-Expressions

⇒ "Extensions" to *Lisp*

- Side Effects (setq, putprop, . . .)
- Numbers
- Dotted-Pair, Association & Property Lists
- *Lisp qua* Procedural Language (i/o, do, ...)

- Issues
  Recursion, Variables, Efficiency
  Funarg Problem (Scoping)
  Program=Data (Eval, nlambda, OOp)
  Lambda Calculus
  SECD machine
- Example (polynomials)

No Side Effects

```
→ (PLUS 11 23)
34
→ (CAR '(A B C))
A
→ (PLUS 11 23)
34
→ (PLUS (TIMES X 2) 5)
29
→ (PLUS 11 23)
34
→ (CAR '(A B C))
A
→ (PLUS (TIMES X 2) 5)
29
        . . .
```


Side-Effect Free — Def'n

- Form σ has NO side effects if
  Evaluating it does not change anything.
- Hence: Value of form τ is the same
  whether or not σ was evaluated
  for any τ.

  → $\underline{\tau}$
  ⟨v1⟩
  → $\underline{\sigma}$
  ⟨v2⟩
  → $\underline{\tau}$
  ⟨v1⟩

- Examples: Any form using only
  PLUS, CAR, CONS, . . .

Functions with Side Effects – SETQ

```
→ my-const
undefined variable ... blah blah blah ...
→ (SETQ my-const '(A B C))
(A B C)
→ my-const
(A B C)
→ (CAR my-const)
A
→ (SETQ my-const '(t 4))
(t 4)
→ my-const
(t 4)
→ (CAR my-const)
t
```

Functions with Side Effects — DEF

```
          ...
→ ( (LAMBDA (X) (CDR X))  '(A B C) )
(B C)
→ ( (LAMBDA (X) (CDR X))  '(Q t) )
(t)
→ (DEF my-fn (LAMBDA (X) (CDR X)) )
my-fn
→ (my-fn  '(A B C))
(B C)
→ (my-fn  '(Q t)
(t)
→ (my-fn (my-fn my-const)))
(C)
→ (DEF my-fn (LAMBDA (X) (CAR X)) )
my-fn
→ (my-fn  '(A B C))
A
→ (my-fn  '(A B C))
A
          ...
```


User-Defined Function with Side Effects

```
          ...
→ (SETQ my-var  5)
5
→ my-var
5
→ (DEF fn2 (LAMBDA (X) (SETQ my-var X)) )
fn2
→ my-var
5
→ (fn2  '(A B C))
(A B C)
→ my-var
(A B C)
→ (fn2  (LIST (PLUS 3 4)))
(7)
→ my-var
(7)
          ...
```


DEF ⇒ DEFUN

(DEFUN name (v_1 ... v_n) ⟨form⟩)

is an ABBREVIATION for

(DEF name (LAMBDA (v_1 ... v_n) ⟨form⟩))

The SETQ "Function"

SETQ does NOT evaluate its first argument.

```
→ B
(t t)     ;; Hence B's value is (t t)
→ X
undefined   blah ... blah... blah...
→ (setq X 'B)
B     ;; Notice:  NOT an error.  X is NOT evaluated.
→ X
B     ;; Notice:  X now has a value.
→ B
(t t)    ;; but B's value has not changed.
→ (setq X 19)
19
→ X
19       ;; Notice:  X's value has changed,
→ B
(t t)       ;; but B still have the same value.
→ (setq X (+ 4 5))
7       ;; Notice SETQ does evaluation its 2nd arg.
```

The SET Function

SET DOES evaluate its first argument.

```
→ B
(t t)     ;; Hence B's value is (t t)
→ X
undefined   blah ... blah... blah...
→ (set X '(foo bar))
undefined   blah ... blah... blah...
      ;; Notice:  This IS an error, as X is evaluated.
→ (setq X 'B)
B      ;; Notice:  X now has the value:  B.
→ X
B
→ (set X (+ 100 12))
112
→ X
B        ;; But it wasn't X whose value changed!
→ B
112       ;; It was B's!
→ (set (car '(a b)) 6)
6
→ a
6
```