Numbers in Lisp

- *Numbers* are special atoms.
  (Each evaluates to itself.)

  → 5
  5

  → (list 5 'a)
  (5 a)

- Numberp tests whether an s-expr is a numeric atom.

  Numberp of a is $\begin{cases} \texttt{t} & \text{if a is a number} \\ \texttt{nil} & \text{otherwise} \end{cases}$

  → (numberp 12)
  t

  → (setq n 25)
  25

  → (numberp 'a)
  nil

  → (numberp n)
  t

  → (numberp '(1 2))
  nil

Numerical Operations

– n-ary Functions: + PLUS * TIMES ...
– Binary Functions: - MINUS / DIVIDES MOD
– Unary Functions: ABS SIN COS ...
– Binary Predicates: < > ...
– Unary Predicates: ZEROP ...

Use prefix notation:

→ <u>(* 15 20)</u>
300
→ <u>(+ 1 2 3 4 5)</u>
15
→ <u>(mod 20 8)</u>
4
→ <u>(+ (* 2 3) (- 9 5))</u>
10
→ <u>(> (+ 4 6) (mod 23 4))</u>
t

"Dot Notation"

- CONS can (really) take ANY pair of S-Exprs

  (Earlier, dealt only with S-expr and <u>List</u>.)

- Value of (CONS s_1 s_2) is (s_1 . s_2)
  For any s_1, $s_2 \in$ s-expr.

→ <u>(CONS 'a 'b)</u>
(a . b)

→ <u>(CONS 4 '(a b c))</u>
(4 . (a b c))

→ <u>(CONS '(t) '(a . b))</u>
((t) . (a . b))

→ <u>(CONS 4 '(a . b))</u>
(4 . (a . b))

- Retrieving components:
  CAR of (a . b) returns a
  CDR of (a . b) returns b

→ <u>(CAR '(a . b))</u>
a

→ <u>(CDR (CONS 'a 'b))</u>
b

→ <u>(CDR '((t) . (a . b)))</u>
(a . b)

→ <u>(CAR (CDR '(4 . (a . b))))</u>
a

Notation (Equivalence)

- Can write (s_1 . (s_2 $\cdots$))
  as (s_1 s_2 $\cdots$)

  Hence: (a . (b . c)) $\leadsto$ (a b . c)

- Can write (s_1 s_2 $\cdots$ s_n . nil)
  as (s_1 $s_2 \cdots$ s_n)

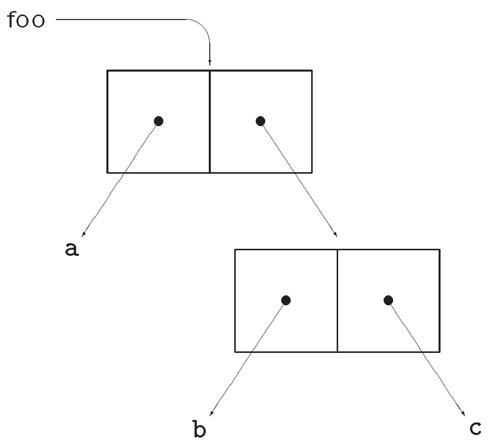
  Hence: (CONS 'a nil) $\leadsto$ (a . nil) $\leadsto$ (a)

- Notice:
  When CONS's 2^{nd} arg is list,
  AS BEFORE!!

Dotted Pair – Internals

→ (SETQ foo (CONS 'a (CONS 'b 'c)))

(a . (b . c))



Association Lists

- An **Association List** is a list of DOTTED-pairs.

  `CAR` of each DOTTED-pair is attribute
  `CDR` of each DOTTED-pair is value.

- Eg:

```
( (name      . (Bart Selman))
  (hair      . black)
  (children  . ( (Mary Louise)
                 (Jean Pierre) ) )
  (habits    . nil) )
```

- Can be assigned:

```
(setq bart '( (name Bart Selman) (hair .  black)
              (children (Mary Louise) (Jean Pierre) )
              (habits) ) )
```


`ASSOC` Function

- `ASSOC` takes two arguments
  Attribute (an atom)
  Alist (an association list)

  returns entire Dotted-Pair if match is found.

- Using `bart`

```
→ (ASSOC 'name bart)
(name Bart Selman)

→ (ASSOC 'children bart)
(children (Mary Louise) (Jean Pierre))

→ (ASSOC 'habit bart)
(habit)

→ (ASSOC 'mother bart)
nil

→ (CDR (ASSOC 'name bart))
(Bart Selman)
```

- Requires $2n$ `CONS`-cells overhead

Property Lists

- An atom can have
  - a value (via `SETQ`)
  - a function (via `DEFUN`)
  - ⇒ a property list (via `PUTPROP`)

- Eg, atom `bart` can have

attribute name	attribute values
name	(Bart Selman)
hair	black
children	((Mary Louise) (Jean Pierre))
habits	nil

Property List Operations

- (get ⟨atom⟩ ⟨attribute⟩)
  returns value of ⟨atom⟩'S attribute ⟨attribute⟩.

- (setf (get ⟨atom⟩ ⟨attribute⟩) ⟨value⟩)
  assigns (changes) value of ⟨atom⟩'S
  attribute ⟨attribute⟩ to be ⟨value⟩.

  Eg: after (setf (get 'bart 'habits) 'coffee)
  (get 'bart 'habits) ⤳ coffee.

- (remprop ⟨atom⟩ ⟨attribute⟩)
  removes attribute ⟨attribute⟩ from ⟨atom⟩'S property list.

  Eg: after (remprop 'bart 'habits)
  (get 'bart 'habits) ⤳ nil.

Use of Properties

→ (setf bart 7)
7
→ (setf (get 'bart 'name) '(Bart Selman))
(Bart Selman)
→ (setf (get 'bart 'hair) 'black)
black
→ (get 'bart 'name)
(Bart Selman)
→ (get 'bart 'mother)
nil
→ (remprop 'bart 'name)
T
→ (get 'bart 'name)
nil
→ bart
7

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) · · · (p_n q_n))

Other stuff?

- Typical *Lisp* system have many

  functions: NULL, EQUAL, MEMQ, APPEND, . . .

  special forms: LIST, AND, OR, . . .

  constants: (numbers), . . .

  just for CONVENIENCE!

  EG: FranzLisp have > 700 of them

- "Impurities"
  setq, setf
  DEF (+ new functions it allows)

- (CADAR e)
  means (CAR (CDR (CAR e)))
  (CADAR '((A B) (C D))) ⤳ B