Data Types

- "Symbols"

  `[ abc a8 t nil abc.$4 ...]`

- Numbers

  Integer `[ 43 -99 ...]`

  Real `[ 22.3 -9.9 ...]`

  "Bignums" `[ 439939482 -94322232 ...]`

- Strings

  `["abc" "84 . a%3" ...]`

- Hunks, . . .

(Any can be "atom" of s-expression)

I/O in Lisp – Examples

```
→ (car (setq x (read)))
(a b c)          ; user types in
a                ; value of form.

→ x
(a b c)
→ (print 's)
snil             ; (print 's)  prints ``s'', then returns ``nil
→ (print (+ 3 4))
7nil
→ (null (print (+ 3 4)))
7t
→ (defun test () (print (sqrt (read))) (terpri))
test
→ (test)
16               ; user types in
4.0
nil
→
```


I/O in Lisp – Def'n

`(read)` – reads an s-expr from terminal;
returns s-expr as `(read)`'s value.
[No arguments]

`(print` ⟨`s-expr`⟩`)` – prints s-expr ⟨`s-expr`⟩ on terminal; returns `nil`.
[⟨`s-expr`⟩ is any s-expression]

`(terpri)` – flushs buffer, prints carriage return.
[No arguments]

`(load` ⟨`file`⟩`)` – loads file named ⟨`file`⟩.
[⟨`file`⟩ is a string (file name)]

Functions for Side-Effects

(progn ⟨$\texttt{form}_1$⟩ ⟨$\texttt{form}_2$⟩ $\cdots$ ⟨$\texttt{form}_m$⟩)

Evaluates all forms, ⟨$\texttt{form}_i$⟩ $(i = 1..m)$ in order.

Returns value of final form, ⟨$\texttt{form}_m$⟩
(Ignores other values)

Takes ANY number of forms

(prog2 ⟨$\texttt{form}_1$⟩ ⟨$\texttt{form}_2$⟩ $\cdots$ ⟨$\texttt{form}_m$⟩)

Like `progn`, but returns 2^{nd} value, not last.

Evaluates all forms, ⟨$\texttt{form}_i$⟩ $(i = 1..m)$ in order.

Returns value of 2^{nd} form, ⟨$\texttt{form}_2$⟩
(Ignores other values)

Takes ANY number of forms

Truth about `LAMBDA`

FranzLisp accepts:

(LAMBDA (a_1 $\cdots$ a_n) $\langle$form$_1\rangle$ $\cdots$ $\langle$form$_m\rangle$)

ie, λ evaluate *any number of forms*
returning value of final form, $\langle$form$_m\rangle$.

(Earlier, insisted on exactly 1 form.)

```
→ ( (LAMBDA (a) (print a) (setq x a) (+ a 3))  19)
19        ; printed by ''(print a)'' statement.
22        ; value of this form.
→ x
19        ; x's value has been effected.
→
```


Truth about `COND`

Earlier, insisted that each COND "clause"
take exactly 2 forms.
but, ...can take any number, from 1 on.

(COND ($\langle q_1^1\rangle$ $\langle q_2^1\rangle$ $\cdots$ $\langle q_{m1}^1\rangle$)
($\langle q_1^2\rangle$ $\langle q_2^2\rangle$ $\cdots$ $\langle q_{m2}^2\rangle$)
...
($\langle q_1^n\rangle$ $\langle q_2^n\rangle$ $\cdots$ $\langle q_{mn}^n\rangle$))

where each $\langle q_j^i\rangle$ is a form, $mi \in \mathcal{Z}^+$.

If $\langle q_1^i\rangle$ is nonNIL,
Then evaluate $\langle q_j^i\rangle$ forms, for $j = 2..mi$.
Return value for final form $\langle q_{mi}^i\rangle$.

If $mi = 1$, and if $\langle q_1^i\rangle$ is nonNIL,
then return $\langle q_1^i\rangle$'s value.

Example of (real) `COND`

```
→ (defun swp (y)
     (COND ( x )
           ( y (print ''x was nil, is now '')
               (print (setq x y))
               (terpri)   7) ) )
→ (setq x 'fred)
fred
→ (swp 18)
fred             ; just prints out value of x.
→ (setq x nil)
nil
→ (swp 18)
x was nil, is now 18     ; prints msg, and resets x.
7                ; value that form returns.
→ x
18
→ (setq x nil)
nil
→ (swp nil)
nil              ; COND fails.
→
```


Iteration in Lisp – Example

Given $n \in \mathcal{Z}^+$,
compute list (0 1 2 $\cdots$ n).
As side effect: print out (descending) numbers.

```
(defun printer (n)
  (do  ( (i     n    (1- i))
         (ans  nil  (cons i ans)) )
       ((zerop i) ans)
       (print i)
       (terpri) ) )
```

DO Form – In General

```
(do  ( (v1   iv1  sv1)
       (v2   iv2  sv2)
        ...
       (vk   ivk  svk) )
     (⟨Rform0⟩ ⟨Rform1⟩ ··· ⟨Rformm⟩)
     ⟨Bform1⟩ ··· ⟨Bformp⟩ )
```

---

where

v_i are variables,

iv_i, sv_i are forms,

$\langle Rform_j \rangle$, $\langle Bform_k \rangle$ are forms.

Evaluating DO Form

```
(do  ( (v1   iv1  sv1)
       (v2   iv2  sv2)
        ...
       (vk   ivk  svk) )
     (⟨Rform0⟩ ⟨Rform1⟩ ··· ⟨Rformm⟩)
     ⟨Bform1⟩ ··· ⟨Bformp⟩ )
```

1. Each v_i initialized to iv_i.

2. If $\langle Rform_0 \rangle$ evaluates to nonNIL.
Then DO evaluates other $\langle Rform_i \rangle$ forms, in order;
returns value of last one – $\langle Rform_m \rangle$.

3. Otherwise [if $\langle Rform_0 \rangle$ evaluates to NIL],
Evaluate each of the $\langle Bform_j \rangle$ forms, in order.
If any of form (return ⟨form⟩),
Then value of overall do loop is value of ⟨form⟩.

Otherwise, just cycle through:

4. v_i's new value becomes sv_i.

[All sv_i's evaluated at same time,
so must use *current* value of variables.]

Repeat $\langle Rform_0 \rangle$ test, etc.

If sv_i omitted: v_i stays same
unless changed by setq, . . . in body or elsewhere.

If iv_i omitted: v_i begins without value.

If only 1 form, $\langle Rform_0 \rangle$, its nonNIL value is returned.