19 Language Exploration : LISP

Friday, December 1, 2023 11:12 AM

Language Exploration

- FUNDAMENTALS OF FUNCTIONAL PROGRAMMING.
  1.- No assignment. (Just labels)
  2.- Computation is performed by "function composition"
     - little or no sequential execution.
     - Recursion over iteration. $s(h(g(f(x))))$
  3.- Functions are "First class" entities.
     - can be passed as arguments to functions
     - can be returned as return values of functions.

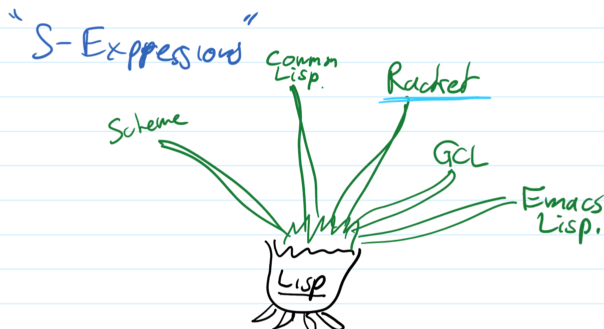
- LISP
  - History 1960 by John McCarthy

He was working on a "Model of computation" inspired by Alonso Church's "lambda Calculus"

$f(x)$ $f.x$

$\lambda.x$ ← anonymous function that takes one parameter x

"S-Expressions"



- The Core of Lisp:

exceptions: ; # '

Syntax

atom :- a sequence of letters or numbers or some symbols

e.g. foo apple 123 cat2 2cat

the_snake hello? a-b a+b

expressions - an atom

or

- a list of zero or more expressions.
  - seperated by spaces
  - enclosed in parenthesis.

e.g. foo (apple cat2) ()

(a b c) (apple)

(foo (bar) (apple 123))

Semantics: idea:

(a b c d) → data.

→ function apply function a to arguments b c d

- BASIC FORMS (pre-defined functions)
  - (quote x) evaluates to x

e.g.

(quote a) → a

shorthand 'a → a

(quote (a b c)) → (a b c)

'(a b c) → (a b c)

- Special atoms

t - true

nil } false
() }

- (atom x) evaluates to t if x is an atom
  nil otherwise.

e.g. (atom apple) ⇒ ⊗ apple is undefined
(atom 'apple) ⇒ t
(atom '(a b c)) ⇒ nil
(atom (a b c)) ⇒ ⊗ a is undefined.
(atom (atom 'a)) ⇒ (atom 't) ⇒ t
(atom '(atom 'a)) ⇒ nil

data
do not evaluate

- (eq x y) evaluates to t if both x and y are atoms and are the same atom, or both the empty list. Nil otherwise.

e.g. (eq 'a 'a) ⇒ t
(eq 'a 'b) ⇒ nil
(eq 't (atom 'a)) ⇒ (eq 't 't) = t

- (car l) expects l to be a list and evaluates to the first element in list l

e.g.
(car '(a b c)) ⇒ a
(car '(apple banana)) ⇒ apple
(atom (car '(a b c))) ⇒ (atom 'a) ⇒ t

- (cdr l) expects l to be a list and evaluates to the list after the first element of l

(cdr '(a b c)) ⇒ (b c)
(cdr '(a)) ⇒ ()
(cdr (cdr '(a b c))) ⇒ (cdr '(b c)) ⇒ (c)

- (cons x y) expects y to be a list, and evaluates to the list that consists of x followed by y

(cons 'a '(b c)) ⇒ (a b c)
(cons '(b c) '(d e)) ⇒ ((b c) d e)

- Conditional form:

  (cond (p_1 e_1)(p_2 e_2)(p_3 e_3) .... (p_n e_n))

  - the p expressions are evaluated in order until one evaluates to t, then the corresponding e expression is evaluated, and its result is the value of the whole cond expression.
  - evaluates to nil if no p expression evaluates to t.

  e.g. (cond ((eq 'a 'b) 'first)
  ((atom 'a) 'second)
  ((atom 'b) 'third)) ⇒ second

  (in the first clause, (eq 'a 'b) is labelled p and 'first is labelled e)

- Functional Forms
  - (lambda (p_1 p_2 ... p_n) e)

    an anonymous function with parameters $p_1 \ldots p_n$ and expression e as its body

    eg. (lambda (a b c) (cons a (cons b (cons c nil))))

  - (funcall f (p_1 p_2 ... p_n)) expects f to be a function, evaluates to f applied to arguments $p_1 \ldots p_n$

    e.g. (funcall (lambda (a b c)
    (cons a (cons b (cons c nil))))
    ('apple 'banana 'orange))

    ⇒ (apple banana orange)

  - (defun foo (p_1 p_2 -- p_n) e) defines a named function foo with parameters $p_1 \ldots p_n$ and body e.

    e.g.

    (defun make-triple (a b c)
    (cons a (cons b (cons c nil))))

(make-triple 'apple 'orange 'banana) ⇒ (apple orange banana)

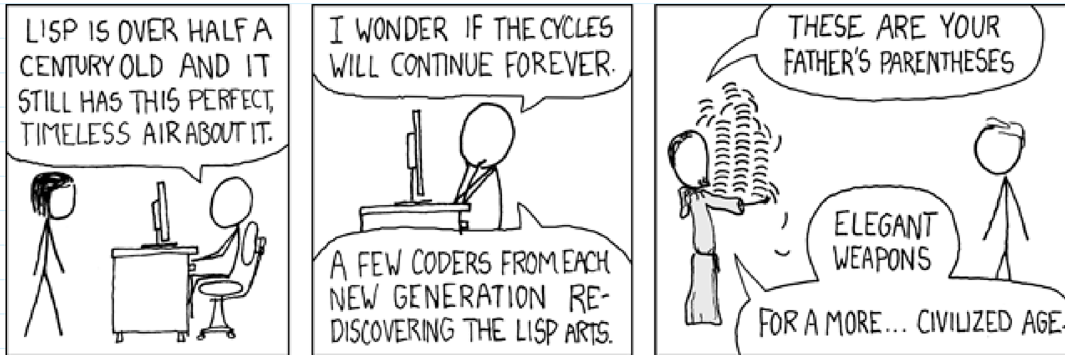
more example functions

```
(defun is-null? (x)
    (eq x nil))

(defun and (x y)
  (cond (x (cond (y t)
                 (t nil)))
        (t nil) ))

(defun append (l x)
  (cond (eq l '()) (cons x '()))
        (t (cons (car l)
                 (append (cdr l) x))))
```

[h · ··· ··· ·] x ⇒ [h ··· x]

(l labels the list [h · ··· ·])

h —cons— [······ x]



Demo !!