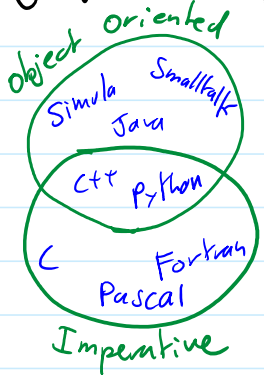
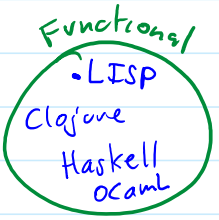
Language Exploration : Lisp

Monday, April 21, 2025 12:22 PM

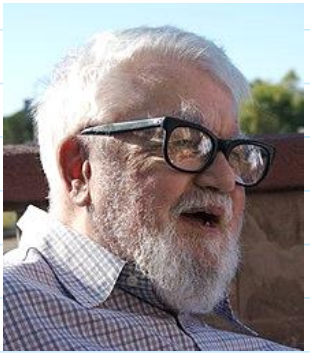
Language Exploration





- FUNCTIONAL PROGRAMMING
  1. No assignment. (Just labels)
  2. Computation is performed by "function Composition" $h(f(g(x)))$
     - No sequential execution
     - No iteration, Recursion
  3. Functions are "first-class"
     - can be passed as arguments to functions.
     - can be return values from functions.

- LISP
  List Processor.
  1960's by John McCarthy



- While working in the "lambda calculus" of Alonzo Church.
  concept:- Anonymous functions.

  $f(x)$
  $\sin(y)$

  $f.x$ $\lambda.x$
  $\sin.y$ $\lambda.y$

- The CORE of LISP

Syntax:

- atom:- a sequence of letters, numbers or symbols (ex: ; # ' `)

  e.g. foo apple 123 cat2 the_snake is_even? a-b

- expressions:- -an atom
  or
  -a list of zero or more expressions separated by spaces enclosed in parenthesis.

  e.g foo (cat2 123) ()
  (a b c) (foo (bar) (apple 123))
  (apple)

Semantics.

(a b c d) → data
→ function application
apply function a to parameters b c d

e.g: (sin 4.57)
(fibo 7) (gcd 37 27)

- Basic Forms (built in functions)
  - (quote x) evaluates to x

e.g
(quote a) ⇒ a
'a ⇒ a
(quote (a b c)) ⇒ (a b c)
'(a b c)

- Special atoms.

t - true
nil } false
() }

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

  eg. (atom 'apple) ⇒ t
  (atom '(a b c)) ⇒ nil
  (atom (a b c)) ⊗ unknown function a
  (atom (atom 'a)) ⇒ (atom 't) ⇒ t
  (atom '(atom 'a)) ⇒ nil
  (atom 'nil) ⇒ t

- (eq x y) :- t if x and y evaluates to the same
  atom or both the empty list.

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

- (car x) :- expects x to be a list
  evaluates to the first element of x

  eg. (car '(a b c)) ⇒ a
  (atom (car ((eq 'a 'b) nil))) ⇒
  (atom (car (nil nil))) ⇒
  (atom nil) ⇒
  t

- (cdr x) : expects x to be a list
  evaluates to the list of elements after
  the first element of x

  e.g. (cdr '(a b c)) ⇒ (b c)
  (cdr '(a)) ⇒ ()
  (cdr (cdr '(a b c))) ⇒ (c)

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

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

- Conditional form:

(cond (p1 e1) (p2 e2) (p3 e3) ... (pn en))

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

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

shorthand (if p e1 e2) (cond (p e1)
(t e2))

- Functional Forms.
  - (lambda (a1 a2 a3 .. an) e)
    an anonymous function with parameters a1, a2 .. an and expression e as its body

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

  - (funcall f (a1 a2 .. an)) expects f to be a function
    applies f to parameters a1 .. an

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

- (defun foo (a1 a2 ... an) e)
  defines a named function with name foo and parameters a1 ... an and body e.

and parameters $a_1 \ldots a_n$ and body e.

e.g.

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

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

- (eval e) evaluates expression e.

- Special Atoms / Functions

numbers are understood as themselves.
+ - * / mod div.

```
(+ 5 3 7)
```

- Demo Functions
  - is-null?
  - Fizz buzz

3, not 5 Fizz
5, not 3 Buzz
both Fizzbuzz.

```
(+ 1 5 7)
```

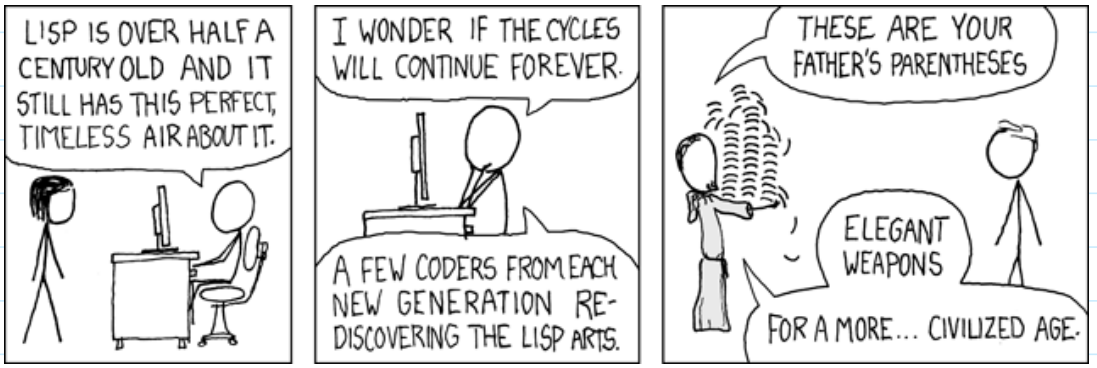
```
(defun mysum. (L) ~~~ )
(defun mycount (x L) ~~~ )
```

L
() ⇒ nil
(x ~~~) ⇒ 1 + (mycount x cdr (L))

() ⇒ nil

(x ~~~) ⇒ 1 + (mycount x cdr(L))

(y ~~~) ⇒ (mycount x cdr(L))



n is even { true λ.n n/2 ; False λ.n n*3+1

31 94 47 142 71 1

1 = (1)