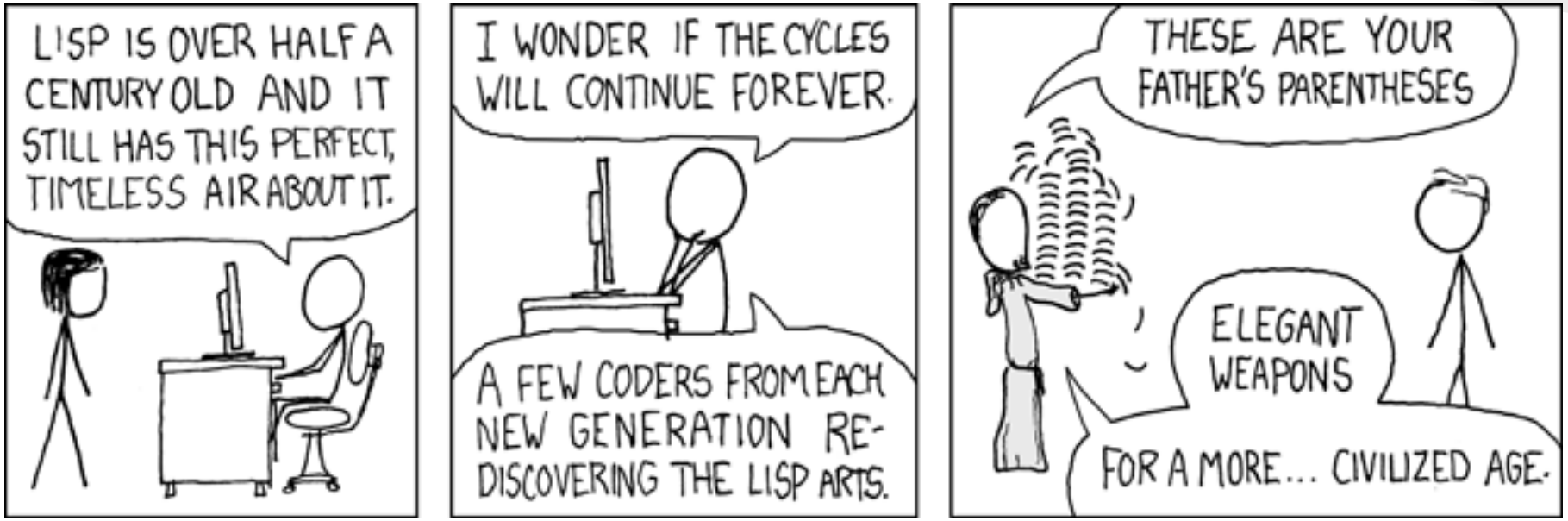
LISP

VinterLAN 2015
Anders Sikvall

Vad är LISP?


LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.
I WONDER IF THE CYCLES WILL CONTINUE FOREVER.
A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.
THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

List Processor

- En LISP-maskin är en list-processor
- En lista är ett objekt som innehåller 0 eller flera andra objekt
- I LISP är allt ett objekt…
- Listor kan alltså innehålla vad som helst egentligen hela program

Exempel: Addera två tal

```
(+ 10 20)
```

Ovanstående är en lista som innehåller 3 st element. Varje element är en atom. Första elementet i en lista är speciellt, det antas vara en funktion.

Polsk notation!

Exempel: Addera två tal generellt

Programexempel

```
(defun addera-tva-tal (tal1 tal2)
  (+ tal1 tal2))
```

Programmet ovan anropas som

```
(addera-tva-tal 10 20)
```

Predikat

Används för att testa om något är sant eller falskt.
zerop, plusp, minusp, listp, evenp, oddp…

```
(if (zerop kalle-kamel)
  (do-if-true)
  (do-if-false))
```

nil är falskt
allt som inte är nil är sant
t är explicit sant

Villkor

```
(if predikat
  (körs om sant)
  (körs om nil))
```

Klassisk if-sats.

Villkor

```
(cond
  (predikat1 resultat1)
  (predikat2 resultat2)
  …
  (t default))
```

Kan jämföras med “case” i C

Exempel: Addera n tal

```
(defun addera-n-tal (tallista)
  (when tallista
    (+ (car tallista) (addera-n-tal (cdr tallista)))))
```

Rekursivitet! Quoted List!

```
(addera-n-tal '(1 2 10 17 22 18 90))
(setq resultat (addera-n-tal '(10 20 30 40 100 200)))
```

car / cdr

car: Ger dig första elementet i en lista

cdr: Ger dig resten av listan förutom första elementet

```
(car '(1 2 3))          -> 1
(cdr '(1 2 3))          -> (2 3)
(cadr '(1 2 3))         -> (2)
(car (cadr '(1 2 3))) -> 2
```

Listor

Enklaste listan är ingen lista. Alls. Nada. Zilch. Null.

```
(nil)
```

Listor byggs av cons-par

```
(a . b)
```

```
Vänsterledet: (car (a . b)) -> a
Högerledet:   (cdr (a . b)) -> b
```

Listor II

Ett element

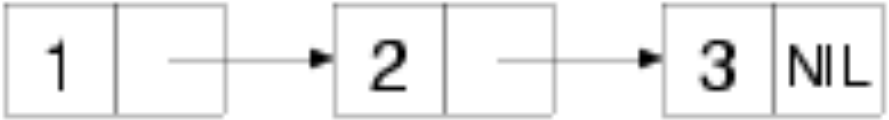
```
(1) -> eg (1 . nil)
(car '(1 . nil)) -> 1
(cdr '(1 . nil)) -> nil
```

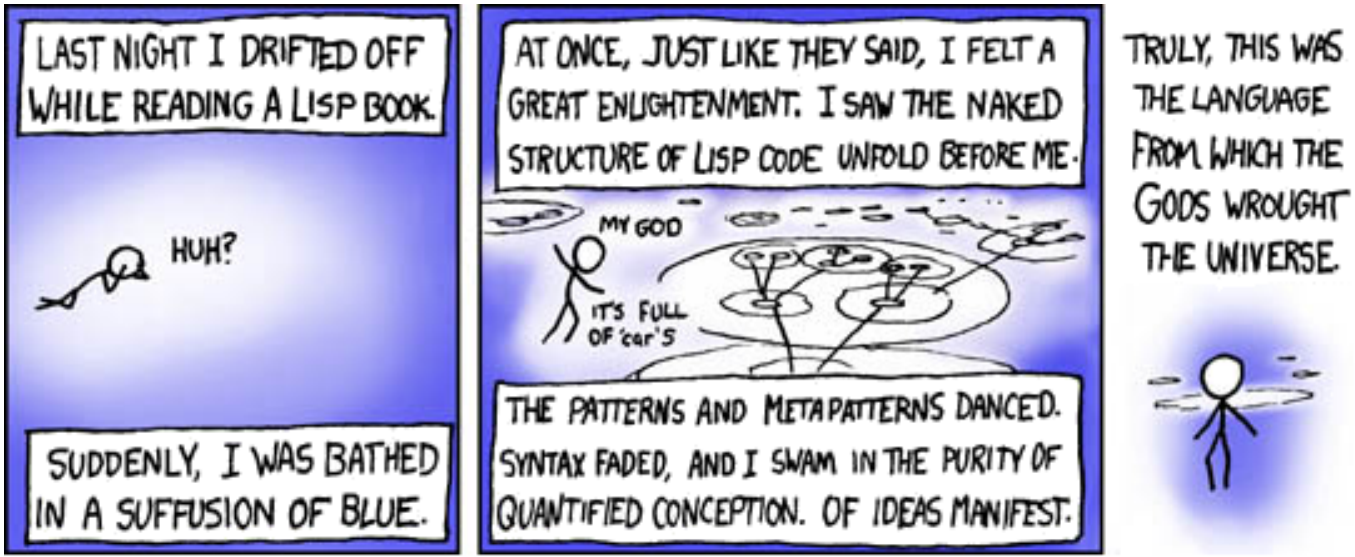
Två element

```
(1 2) -> (1 . (2 . nil))
(car '(1 2)) -> 1
(cdr '(1 2)) -> (2)          <--- WAAAT?
```

Listor III

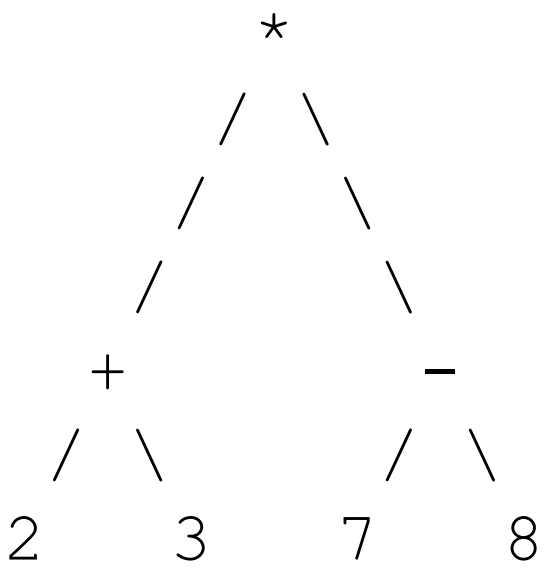
(1 2 3)





Allt är objekt. Även listor.

```
(* (+ (2) (3)) (- (7) (8)))
```



cons

Lägg till element till en lista

```
(cons 1 '(2 3 4))
-> (1 2 3 4)

(cons 'spigg '(gadda aborre laks))
-> (spigg gadda aborre laks)
```

Exempel FSPL

Free space path loss för en lista av distanser från en antenn:

FSPL = 20 log(f) + 20 log(d) - 27.55

```
(defun fspl (frekvens (avstand))
  (when avstand
    (message "%f" (+ (* 20 (log10 frekvens))
                     (* 20 (log10 (car avstand)))
                     -27.55))
    (fspl frekvens (cdr avstand)))

(fspl '(2100 (number-list 100 1000 100)))
```

Exempel: Fibbonaccis talserie

```
(defun fibonacci (N)
  (if (or (zerop N) (= N 1))
      1
    (+ (fibonacci (- N 1)) (fibonacci (- N 2)))))

(defun fibon-recurse (list)
  (when list
    (message "%d" (fibonacci (car list)))
    (fibon-recurse (cdr list))))

(fibon-recurse (number-sequence 1 20 1))
```

Yep, you should be afraid