A primer on lisp

Alán F. Muñoz

2024/06/04

Outline

Introduction

My Computer Science precuel

- On the universality of plain text (2024/01/09)

My Computer Science precuel

- On the universality of plain text (2024/01/09)
- A primer on Lisp (2024/06/04)

My Computer Science precuel

- On the universality of plain text (2024/01/09)
- A primer on Lisp (2024/06/04)
- Emacs: The Editor of a Lifetime (?)

There is no such thing as “the Lisp”

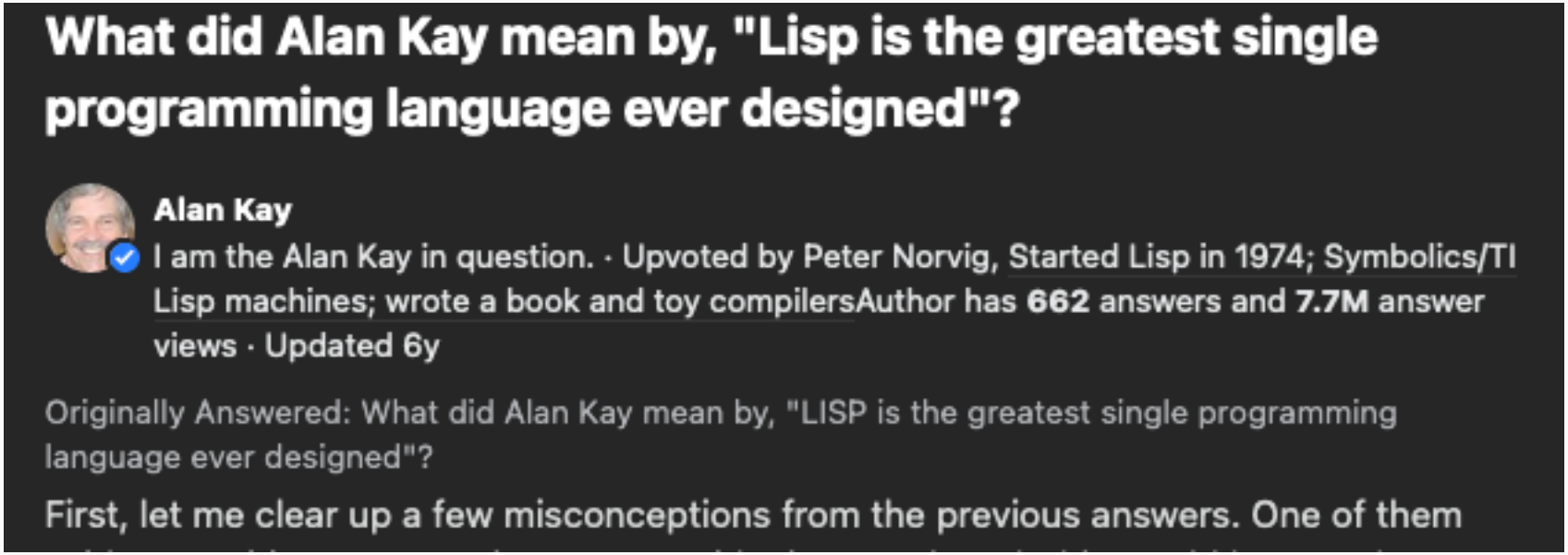
- The name Lisp derives from “List Processing”.
- It is a family of functional programming languages (1958).
- Influenced by lambda calculus (Mathematical abstractions of computation).

Why?

One of the motivations was that he (John McCarthy) wanted something like "Mathematical Physics" — he called it a "Mathematical Theory of Computation". Another was that he needed a very general kind of language to make a user interface AI — called "The Advice Taker" — that he had thought up in the late 50s.

Alan Kay (@Quora), 2018

Literally Alan Kay answered a question about him



What did Alan Kay mean by, "Lisp is the greatest single programming language ever designed"?

Alan Kay
I am the Alan Kay in question. · Upvoted by Peter Norvig, Started Lisp in 1974; Symbolics/TI Lisp machines; wrote a book and toy compilersAuthor has **662** answers and **7.7M** answer views · Updated 6y

Originally Answered: What did Alan Kay mean by, "LISP is the greatest single programming language ever designed"?

First, let me clear up a few misconceptions from the previous answers. One of them

Bits of history

- Designed by John McCarthy (1958); Implemented by Steve Rusell.
- It was tightly coupled to (OG) Artificial Inteligence
- The Common Lisp specification was developed in 1984

The original paper is really old

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

JOHN MCCARTHY, *Massachusetts Institute of Technology, Cambridge, Mass.*

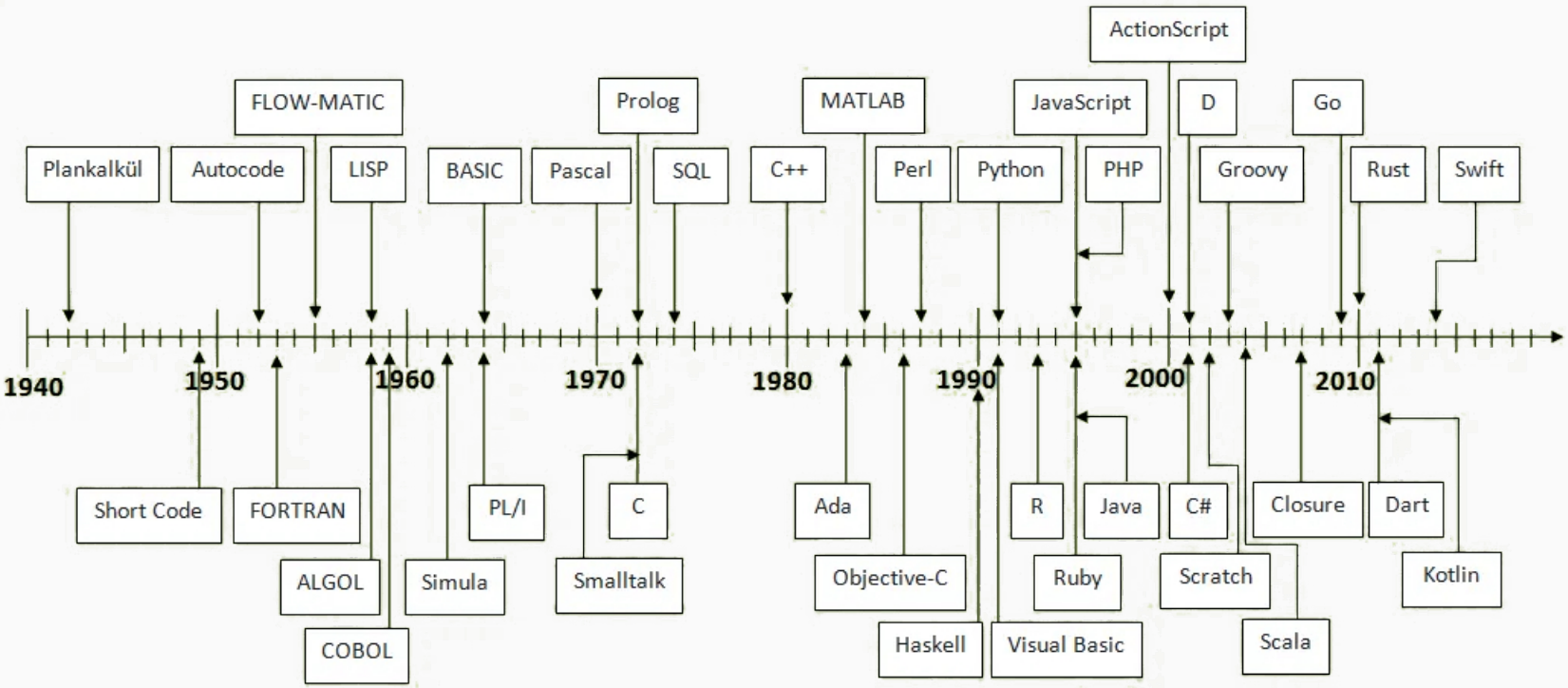
1. Introduction

A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system

2. Functions and Function Definitions

We shall need a number of mathematical ideas and notations concerning functions in general. Most of the ideas are well known, but the notion of *conditional expression* is believed to be new, and the use of conditional

To put the 50s into context



On Lisp simplicity

McCarthy built Lisp out of parts so fundamental that it is hard to say whether he invented it or discovered it.

Sinclair Target, 2018

What were its main innovations?

- Everything (vars, fns) is available at any time

What were its main innovations?

- Everything (vars, fns) is available at any time
- Pervasive interactivity (first)

What were its main innovations?

- Everything (vars, fns) is available at any time
- Pervasive interactivity (first)
- Garbage collection (first)

But its biggest feature is homoiconicity

The code and its data are the same, thus we can process Lisp data with Lisp.

Minor branching: homoicoicity

Python

Data:

- `[]` lists
- `{}` dicts/sets

Code:

- `def` functions
- `class` classes

Example:

```
[1, 2, 3]
```

1 2 3

Lisp

Data:

- `()` all lists

Code:

- `()`
  functions/classes

Example:

```
'(1 2 3)
```

```
(1 2 3)
```

So, code is data and data is code.

- We can then write code that writes code.
- Modifying underlying code (even at compilation is quite easy)

Macros (not lisp-exclusive) are “functions” that write functions whose inputs can be adjusted at compilation.

It is hard to explain why this matters

"Then what is it that makes Lisp so hard to understand? ... Metaprogramming, code and data in one representation, self-modifying programs, domain specific mini-languages, none of the explanations for these concepts referenced familiar territory. How could I expect anyone to understand them!"

Slava Akhmechet, 2006

I will play the enlightenment card

“Lisp is worth learning for the profound enlightenment experience you will have when you finally get it; that experience will make you a better programmer for the rest of your days, even if you never actually use Lisp itself a lot.” I never understood this statement. I never believed it could be true. And finally, after all the pain, it made sense!

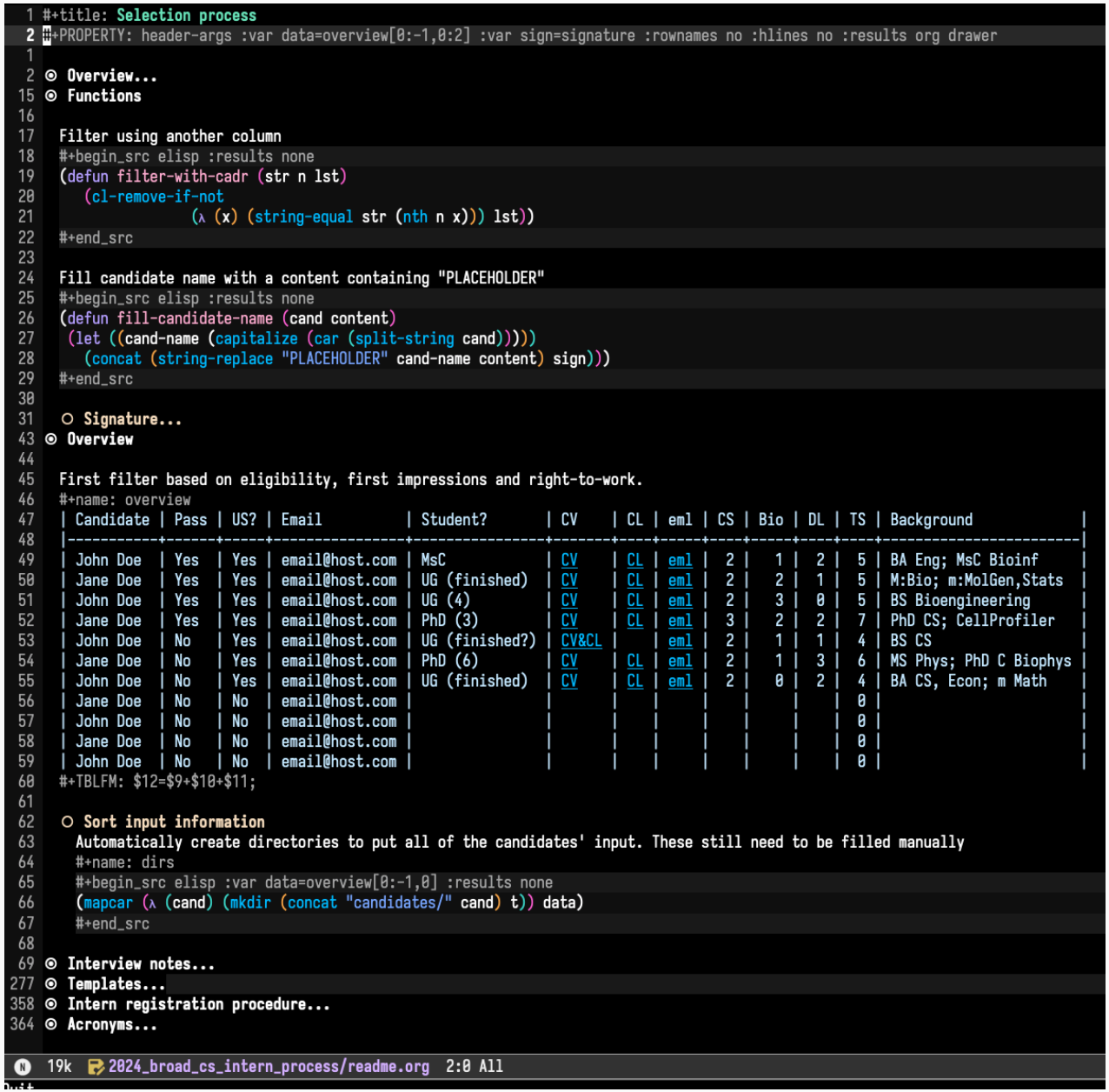
The same person as before, a few paragraphs later

Use-cases

What are the day-to-day use-cases?

- Prototyping.
- Replacing unruly bash scripts.
- Files+Math processing to avoid python.
- Extending Emacs.
- Compiling files ot other languages (e.g., Fennel->Lua)
- Manage your operative system (Guile in Guix)
- Just learning one of the most influential languages and see its ideas elsewhere

Integrates perfectly with reproducible org-mode notebooks



```
#+title: Selection process
#+PROPERTY: header-args :var data=overview[0:-1,0:2] :var sign=signature :rownames no :hlines no :results org drawer

⊙ Overview...
⊙ Functions

 Filter using another column
 #+begin_src elisp :results none
 (defun filter-with-cadr (str n lst)
    (cl-remove-if-not
                (λ (x) (string-equal str (nth n x))) lst))
 #+end_src

 Fill candidate name with a content containing "PLACEHOLDER"
 #+begin_src elisp :results none
 (defun fill-candidate-name (cand content)
  (let ((cand-name (capitalize (car (split-string cand)))))
    (concat (string-replace "PLACEHOLDER" cand-name content) sign)))
 #+end_src

 ○ Signature...
⊙ Overview

 First filter based on eligibility, first impressions and right-to-work.
 #+name: overview
 | Candidate | Pass | US? | Email           | Student?       | CV    | CL | eml | CS | Bio | DL | TS | Background             |
 |-----------+------+-----+-----------------+----------------+-------+----+-----+----+-----+----+----+------------------------|
 | John Doe  | Yes  | Yes | email@host.com  | MsC            | CV    | CL | eml |  2 |   1 |  2 |  5 | BA Eng; MsC Bioinf     |
 | Jane Doe  | Yes  | Yes | email@host.com  | UG (finished)  | CV    | CL | eml |  2 |   2 |  1 |  5 | M:Bio; m:MolGen,Stats  |
 | John Doe  | Yes  | Yes | email@host.com  | UG (4)         | CV    | CL | eml |  2 |   3 |  0 |  5 | BS Bioengineering      |
 | Jane Doe  | Yes  | Yes | email@host.com  | PhD (3)        | CV    | CL | eml |  3 |   2 |  2 |  7 | PhD CS; CellProfiler   |
 | John Doe  | No   | Yes | email@host.com  | UG (finished?) | CV&CL |    | eml |  2 |   1 |  1 |  4 | BS CS                  |
 | Jane Doe  | No   | Yes | email@host.com  | PhD (6)        | CV    | CL | eml |  2 |   1 |  3 |  6 | MS Phys; PhD C Biophys |
 | John Doe  | No   | Yes | email@host.com  | UG (finished)  | CV    | CL | eml |  2 |   0 |  2 |  4 | BA CS, Econ; m Math    |
 | Jane Doe  | No   | No  | email@host.com  |                |       |    |     |    |     |    |  0 |                        |
 | John Doe  | No   | No  | email@host.com  |                |       |    |     |    |     |    |  0 |                        |
 | Jane Doe  | No   | No  | email@host.com  |                |       |    |     |    |     |    |  0 |                        |
 | John Doe  | No   | No  | email@host.com  |                |       |    |     |    |     |    |  0 |                        |
 #+TBLFM: $12=$9+$10+$11;

 ○ Sort input information
   Automatically create directories to put all of the candidates' input. These still need to be filled manually
   #+name: dirs
   #+begin_src elisp :var data=overview[0:-1,0] :results none
   (mapcar (λ (cand) (mkdir (concat "candidates/" cand) t)) data)
   #+end_src

⊙ Interview notes...
⊙ Templates...
⊙ Intern registration procedure...
⊙ Acronyms...

N  19k  2024_broad_cs_intern_process/readme.org  2:0 All
```

It is a general purpose language, so pretty much anything

README

For this presentation I needed the logos for lisps

Set urls

```
(require 'url)

(setq image-urls '(
                   ("cl" . "https://upload.wikimedia.org/wikipedia/commons/4/48/Lisp_logo.svg"
                   ("racket". "https://upload.wikimedia.org/wikipedia/commons/thumb/c/c1/Racke
                   ("scheme" . "https://upload.wikimedia.org/wikipedia/commons/thumb/3/39/Lamb
                   ("fennel" . "https://fennel-lang.org/logo.svg")
                   ("clojure" . "https://upload.wikimedia.org/wikipedia/commons/5/5d/Clojure_l
                   ("elisp" . "https://upload.wikimedia.org/wikipedia/commons/0/08/EmacsIcon.s
                   ("Hy" . "https://upload.wikimedia.org/wikipedia/commons/8/87/Hy_Cuddles.png
                   ("Guile" . "https://upload.wikimedia.org/wikipedia/commons/thumb/b/b3/GNU-G
                   ("lisp" . "https://imgs.xkcd.com/comics/lisp.jpg")
                   ("lisp_cycles" . "https://imgs.xkcd.com/comics/lisp_cycles.png")
                   ("timeline". "https://f.hubspotusercontent40.net/hubfs/14516545/TimelineOfP
```

Download all files

```
(mapcar (lambda (x) (let
 ((name (downcase (car x) ))
 (url (cdr x)))
 (url-copy-file url (format "imgs/%s.%s" name (f-ext url)) t))) image-urls)
```

Convert svgs to png

```
(mapcar (lambda (x) (shell-command
(format "inkscape -h 512 imgs/%s -o imgs/%spng" x (string-remove-suffix "svg" x ))))
(directory-files "imgs" nil (rx ".svg" eos)))
```

Who uses Lisp anyways?

Grammarly (CL)

The London Tube (CL)

Walmart (Clojure)

Puppet (and hence the Broad, Clojure)

Nubank (Clojure)

Hacker News

Technical bits

How does it look?

Functions

```
(f arg1 arg2 arg3)
```

Data

```
(list item1 item2 item3)
```

or

```
'(item1 item2 item3)
```

Scheme are a subfamily of lisp dialects

- Minimalism: Replaces it with more expressiveness
- Lexical scope: You can have the same name in nested functions.
- Tail call optimisations: The compiler optimises recurison.
- Higher focus on functional paradigm.

Playground

Basic operations

```
(+ 2 2)
```

```
4
```

Basic operations

```
(+ 2 (+ 1 1))
```

```
4
```

Setting variables

```
(setq a 1)
```

```
1
```

For **FP** fans: let-in structures are the norm.

```
(let  ((a 3)
       (b 2))
   (+ a b))
```

```
5
```

The primitive data type: Cons

```
(setq cons-cell (cons 'rose 'violet) )
cons-cell

(rose . violet)
```

Equivalent to

```
'(rose . violet)

(rose . violet)
```

CAR and CDR access Cons cells

This is how a cons cell looks

```
cons-cell
```

```
(rose . violet)
```


```
(car cons-cell)
```

```
rose
```

```
(cdr cons-cell)
```

```
violet
```

From this primitive we can build lists

Connected cons cells are lists

```
'(rose . (violet . (buttercup)))
```

```
(rose violet buttercup)
```

From this primitive we can build lists

Connected cons cells are lists

```
'(rose . (violet . (buttercup)))
```

```
(rose violet buttercup)
```

Using “List”

```
(list 'rose 'violet 'buttercup)
```

```
(rose violet buttercup)
```

A diagram shows the linked-list structure

```
(list 'rose 'violet 'buttercup)

(rose violet buttercup)
```

Lists

Lists have their own constructor

```
(list 1 3 2 4)
```

```
(1 3 2 4)
```

Equivalent to a quote (`'`)

```
'(1 3 2 4)
```

```
(1 3 2 4)
```

CAR + CDR on lists

In any list, the first item is the value, the second is a link to the next value. Think of them as **`first`** and **`rest`**!

```
(setq my-list '(1 3 2 4))
(car my-list)
```

```
1
```

```
(cdr my-list)
```

```
(3 2 4)
```

C{AD}R functions enable processing lists with recursion.

```
(cadr my-list)
```

```
3
```

```
(caddr my-list)
```

```
2
```

```
(cadddr my-list)
```

```
4
```

Does this all look weird?

“Why on Earth would anyone want to use a language with such horrific syntax?!”

Your average citizen facing lisp for the first time

There is also the (boring) `nth`

```
(nth 2 my-list)
```

```
2
```

Note that it is zero-indexed.

Lambda

```
(lambda (x) (+ 1  x))
```

```
(lambda (x) (+ 1 x))
```

It returns a function that we can use at the start of a list.

```
((lambda (x) (+ 1  x)) 2)
```

```
3
```

or the built-in `1+`

```
(1+ 2)
```

```
3
```

You can use almost any symbol in variable and function names.

maps and filter

```
(mapcar '1+ my-list)
```

```
(2 4 3 5)
```

Two ways of filtering. Dash:

```
(-filter 'math-evenp my-list)
```

```
(2 4)
```

Common Lisp:

```
(cl-remove-if-not 'cl-evenp my-list)
```

```
(2 4)
```

Flow control: If

```
(setq myvar (list 1 2 3 4 5))
(if (eql 3 (nth 1 myvar))
    "The same" "Not the same")

"Not the same"
```

Flow control: When/Unless

```
"Not the same"

(mapcar (lambda (x)  (when (> x 2) x))
        myvar)

(nil nil 3 4 5)

(mapcar (lambda (x)  (unless (> x 2) x))
        myvar)

(1 2 nil nil nil)
```

Function definition

```
(defun has-cat? (text)
  (if (s-contains? "cat" text)
  "Cat!"  "Nae :("))

has-cat?

(has-cat? "giraffe dog ant")

"Nae :("

(has-cat? "shark cat elephant")

"Cat!"
```

How do we know if a function will be evaluated?

These are all (kind-of) equivalent

```
(list (quote has-cat?)
      'has-cat?
      `has-cat?
      #'has-cat?)

(has-cat? has-cat? has-cat? has-cat?)
```

But not all of these

```
'((quote has-cat?)
  'has-cat?
   #'has-cat?)

('has-cat? 'has-cat? #'has-cat?)
```

We can selectively evaluate terms

The backquote (`) is similar to the quote (') but evaluates elements preceded by a comma.

```
(let ((text "kittycat"))
`((has-cat? text)
 '(has-cat? text)
 ,(has-cat? text)))

((has-cat? text) '(has-cat? text) "Cat!")
```

The programmable programming language

What if there was a way to delay evaluating parts of an expression until compilation?

Example: Simplifying math operations

We can use symbols for more expressiveness, and these functions modify the normal `+-*` on the fly.

Insert on left

```
(->  10
     (- 5)
     (+ 7)
     (* 3))

36
```

Insert on right

```
(->>  10
      (- 5)
      (+ 7)
      (* 3))

6
```

Example: Implementing Python's list comprehensions

Task: Extend **elisp** to add support for list comprehension.

```
[x for x in range(10) if not (x % 2)]
```

```
[0, 2, 4, 6, 8]
```

Example: Implementing Python's list comprehensions

```
(defmacro lcomp (expression for var in list conditional conditional-test)
  (let ((result (gensym)))
    ;; the arguments are really code so we can substitute them
    ;; store nil in the unique variable name generated above
    `(let ((,result nil))
       ;; var is a variable name
       ;; list is the list literal we are suppose to iterate over
       (cl-loop for ,var in ,list
            ;; conditional is if or unless
            ,conditional ,conditional-test
            ;; and this is the action from the earlier lisp example
            do (setq ,result (append ,result (list ,expression))))
           ;; return the result
       ,result)))

(defun range (to) (number-sequence 0 (1- to )))

(lcomp x for x in (range 10) if (= (mod x 2) 0))

(0 2 4 6 8)
```

Conclusions

Wrapping up

- Macros are limited by language syntax.
- S-Expressions (S for symbolic) encode as little a syntax as possible.
- Minimal syntax provides maximum flexibility and extensibility

Takeaways

- The tools we use determine how we can tackle problems
- Even as a recreational tool, Lisp offers insights into the nature of Computer Science
- There is expertise in “old” designs. New is not always better.

There is a Lisp for every need

- General purpose, old and reliable? **`Common Lisp`** (Steel Bank Common Lisp - SBCL)

There is a Lisp for every need

- General purpose, old and reliable? **`Common Lisp`** (Steel Bank Common Lisp - SBCL)
- Need access to **`Java`** libraries + Functional? **`Clojure`**

There is a Lisp for every need

- General purpose, old and reliable? **`Common Lisp`** (Steel Bank Common Lisp - SBCL)
- Need access to **`Java`** libraries + Functional? **`Clojure`**
- Learn Programming Language Theory, Build Doman-Specific Languages (DSL)? **`Scheme/Racket`**

There is a Lisp for every need

- General purpose, old and reliable? **`Common Lisp`** (Steel Bank Common Lisp - SBCL)
- Need access to **`Java`** libraries + Functional? **`Clojure`**
- Learn Programming Language Theory, Build Doman-Specific Languages (DSL)? **`Scheme/Racket`**
- More Emacs in your life and/or more life in your Emacs? **`Emacs Lisp`** (elisp)

There is a Lisp for every need

- General purpose, old and reliable? **`Common Lisp`** (Steel Bank Common Lisp - SBCL)
- Need access to **`Java`** libraries + Functional? **`Clojure`**
- Learn Programming Language Theory, Build Doman-Specific Languages (DSL)? **`Scheme/Racket`**
- More Emacs in your life and/or more life in your Emacs? **`Emacs Lisp`** (elisp)
- Nix but in a sensible language? **`Scheme/GNU Guile`**

There is a Lisp for every need

- General purpose, old and reliable? **`Common Lisp`** (Steel Bank Common Lisp - SBCL)
- Need access to **`Java`** libraries + Functional? **`Clojure`**
- Learn Programming Language Theory, Build Doman-Specific Languages (DSL)? **`Scheme/Racket`**
- More Emacs in your life and/or more life in your Emacs? **`Emacs Lisp`** (elisp)
- Nix but in a sensible language? **`Scheme/GNU Guile`**
- Need **`Lua`** configs but miss parentheses? **`Fennel`**

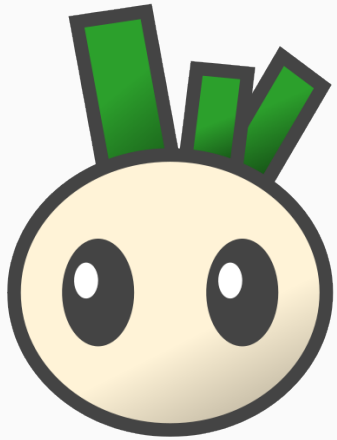
There is a Lisp for every need

- General purpose, old and reliable? **Common Lisp** (Steel Bank Common Lisp - SBCL)
- Need access to **Java** libraries + Functional? **Clojure**
- Learn Programming Language Theory, Build Doman-Specific Languages (DSL)? **Scheme/Racket**
- More Emacs in your life and/or more life in your Emacs? **Emacs Lisp** (elisp)
- Nix but in a sensible language? **Scheme/GNU Guile**
- Need **Lua** configs but miss parentheses? **Fennel**
- Lisp-flavoured **Python**? **Hy**

There is a Lisp for every need

Clojure

Common Lisp



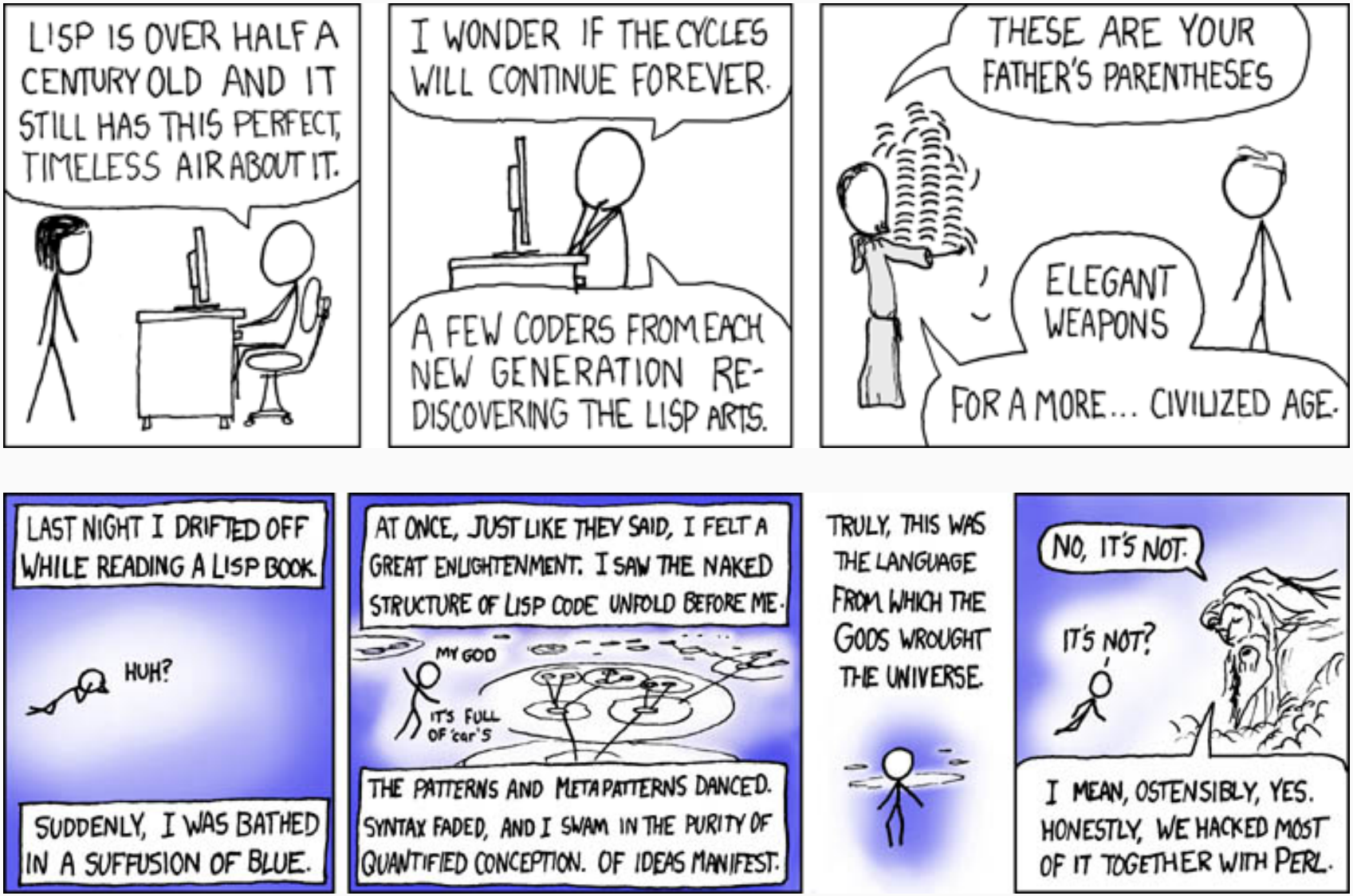
Fennel

Racket

Emacs Lisp

Hy

The land of lisp and other wonders



click me to see the wonders of the world

For those interested

- An intuition for Lisp Syntax (for JS folks)
- The mad lad who implemented comprehensions in Common Lisp
- Peter Seibel's Practical Common Lisp
- Paul Graham's on "Programming bottom up"
- Scheme crash course
- SHRDLU: The (incidentally) first formal example of interactive fiction.
- Is lisp the icing or the cake?
- Data Science in Clojure

https://github.com/scicloj?q=yy&type=all&language=&sort=

References

- This slides: `https://github.com/afermg/2024_06_lisp_primer_CSTutorials/blob/master/slides.pdf`
- `https://ericnormand.me/article/idea-of-lisp`
- Lisp innovations
- Some of the quotes
- Alan's Kay response on Quora