CSCI 334:
Principles of Programming Languages

Lecture 9: LISP

Instructor: Dan Barowy

Williams

Topics

No quiz today
LISP

Your to-dos

1. Lab 4, **due Sunday 3/6** (partner lab)
2. Reading response, **due Wednesday 3/9**. Last one before midterm!

Announcements

- **Midterm exam**, in class, Thursday, March 17.
- **Friday colloquium**, Elena Glassman (Harvard), 2:35pm in Wege Auditorium

"Human-AI (Mis)Communication: challenges and tools for successfully communicating what we want to computers"

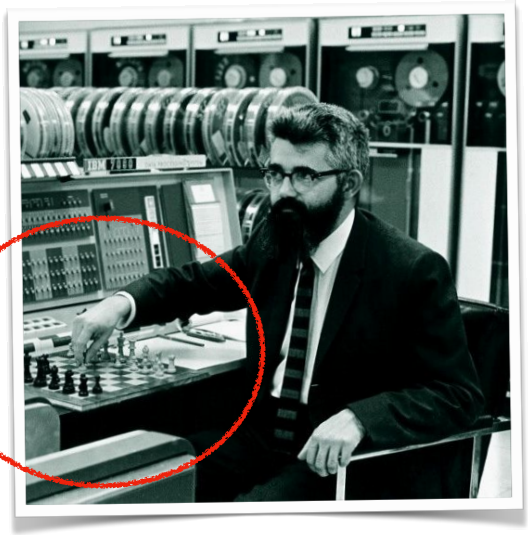
Abstract

While we don't always use words, communicating what we want to a computer, especially an artificially intelligent one, is a conversation—with ourselves as well as with it, a recurring loop with optional steps depending on the complexity of the situation and our request. I will present some key, perhaps previously under-appreciated steps and describe conditions where it is critical to support them, illustrated with examples from recent publications of (1) novel interfaces for interactive program synthesis and (2) interactive visualizations of large piles of complex data. In the process, I will describe relevant theories from the learning sciences, i.e., Variation Theory and Analogical Learning Theory, that have design implications for future interface and interactive system design—to hopefully maximize the bidirectional speed and accuracy of human-AI communication.

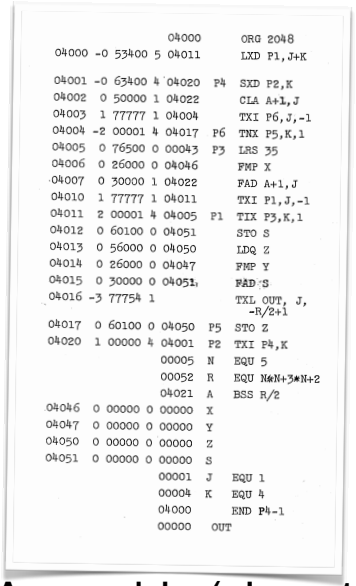
LISP

John McCarthy

IBM 704



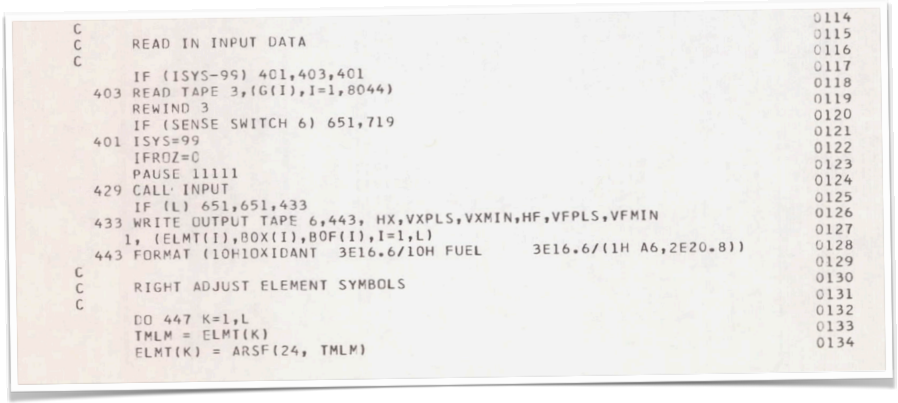
Lisp was invented for AI research



```
                    04000      ORG 2048
04000 -0 53400 5 04011         LXD P1,J+K

04001 -0 63400 4 04020  P4     SXD P2,K
04002  0 50000 1 04022         CLA A+1,J
04003  1 77777 1 04004         TXI P6,J,-1
04004 -2 00001 4 04017  P6     TNX P5,K,1
04005  0 76500 0 00043  P3     LRS 35
04006  0 26000 0 04046         FMP X
04007  0 30000 1 04022         FAD A+1,J
04010  1 77777 1 04011         TXI P1,J,-1
04011  2 00001 4 04005  P1     TIX P3,K,1
04012  0 60100 0 04051         STO S
04013  0 56000 0 04050         LDQ Z
04014  0 26000 0 04047         FMP Y
04015  0 30000 0 04051         FAD S
04016 -3 77754 1               TXL OUT, J,
                                 -R/2+1
04017  0 60100 0 04050  P5     STO Z
04020  1 00000 4 04001  P2     TXI P4,K
                    00005  N   EQU 5
                    00052  R   EQU N*N+3*N+2
                    04021  A   BSS R/2
04046  0 00000 0 00000  X
04047  0 00000 0 00000  Y
04050  0 00000 0 00000  Z
04051  0 00000 0 00000  S
                    00001  J   EQU 1
                    00004  K   EQU 4
                    04000      END P4-1
                    00000  OUT
```

704 Assembly (circa 1954)
(From Coding the MIT-IBM 704 Computer)



```
C                                                                    0114
C     READ IN INPUT DATA                                             0115
C                                                                    0116
      IF (ISYS-99) 401,403,401                                       0117
  403 READ TAPE 3,(G(I),I=1,8044)                                    0118
      REWIND 3                                                       0119
      IF (SENSE SWITCH 6) 651,719                                    0120
  401 ISYS=99                                                        0121
      IFROZ=0                                                        0122
      PAUSE 11111                                                    0123
  429 CALL INPUT                                                     0124
      IF (L) 651,651,433                                             0125
  433 WRITE OUTPUT TAPE 6,443, HX,VXPLS,VXMIN,HF,VFPLS,VFMIN         0126
     1, (ELMT(I),BOX(I),BOF(I),I=1,L)                                0127
  443 FORMAT (10H1OXIDANT  3E16.6/10H FUEL      3E16.6/(1H A6,2E20.8)) 0128
C                                                                    0129
C     RIGHT ADJUST ELEMENT SYMBOLS                                   0130
C                                                                    0131
      DO 447 K=1,L                                                   0132
      TMLM = ELMT(K)                                                 0133
      ELMT(K) = ARSF(24, TMLM)                                       0134
```

FORTRAN (circa 1956)
(From NASA Technical Note D-1737)

```
(defun fact (n)
    (cond ((eq n 0) 1)
          (t (* n (fact (- n 1))))))
```

LISP (circa 1958)

LISP is a “functional” language

- programs are “**mathematical**”
- no **statements**, only **expressions**
- no **mutable variables**, only **declarations**
- therefore, the effect of running a program (“evaluation”) is **purely** the effect of **applying a function to an input**.

Statements vs. expressions

- A **statement** is an operation that changes the state of the computer

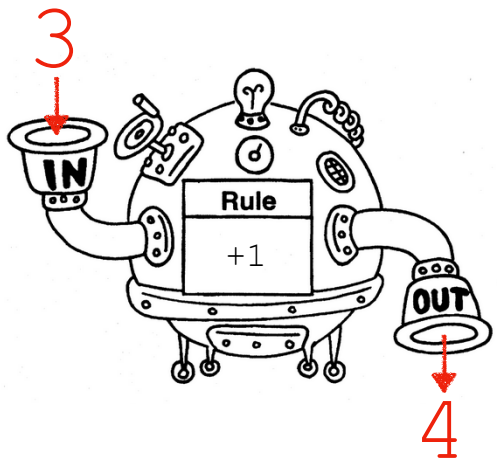
  Java: `i++`

  value stored at location `i` incremented by one
- An **expression** is a combination of values and operations that yields a new value

  Lisp: `(+ i 1)`

  evaluating `+` with `i` and `1` returns `i` plus one
- Lisp **has only expressions**.

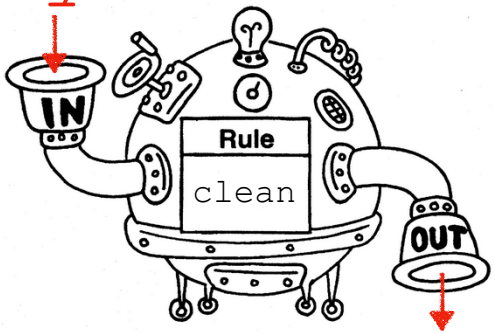
LISP is a "functional" language




```
(defun add-one (n)(+ n 1))
```

LISP is a "functional" language

dirty house

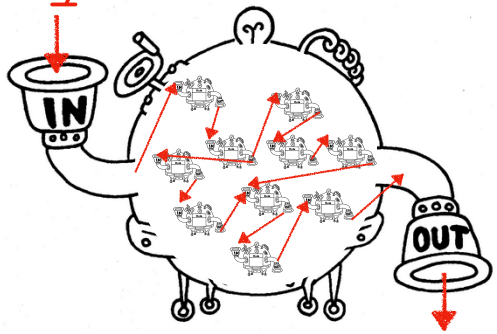


clean house

```
(defun cleaning-robot (dirt) …)
```

Big functions are "composed" of little functions

dirty house

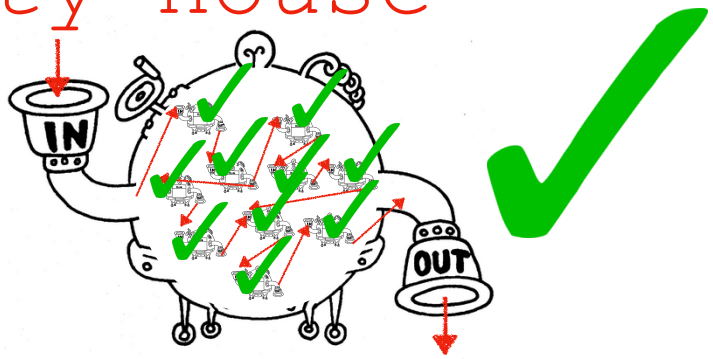


clean house

```
(defun cleaning-robot (dirt) …)
```

Program correctness is easier to achieve

dirty house



clean house

I.e., whole is correct if pieces are correct.

LISP is deeply influenced by the lambda calculus

- all code is either a **value**, a **function definition**, or a **function application**

  value: `1`

  application: `(+ 1 1)`
- syntax is (mind-numbingly) regular

  functions: `(function-name arguments …)`

  values: anything (except `defun`)
- evaluating an expression produces a new value:

  `(+ 1 1)`⇝`2`

REPL
(read-eval-print loop)

Batch mode

Mutable variables

- If you can update a variable in a language, you have **mutable variables**

  Java: `int i = 3;`

  `i = 4;`
- Notice that both lines of code are **statements**
- Lisp **does not have mutable variables**

Immutable variables

- Variables cannot be updated in LISP

  LISP: `(defun my-func (i) …)`

  `(my-func 3)`

  or the shorter

  `((lambda (i) …) 3)`
- Notice that all of the above are expressions
- In fact, functions are the only way to bind values to names in (pure) LISP

Lisp syntax: atoms

- An **atom** is the most basic unit of **data** in Lisp

`4`	Number
`112.75`	Number
`"hello"`	String
`'foo`	Quoted symbol
`t`	Boolean
`nil`	Empty list

LISP control flow

- Conditionals are stated using **`cond`**
  - It's a generalization of **`if`**/**`else`**
  - Think of it as **`switch`** on steroids
  - Syntax:

    `(cond (`p_1 e_1`) ... (`p_n e_n`))`
  - Where p_i is a predicate and

    e_i is an expression to run when p_i is `T`.

`cond`: fibonacci

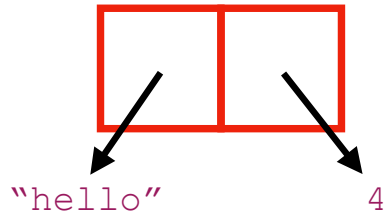
LISP control flow

- LISP has **no loops**
- All **repetition** is done **recursively**, or…
- by using **higher-order functions** (next class!)

Lisp syntax: data structure

- Historically, Lisp has exactly one data structure: the **cons cell**.
- The "cons cell" allows "composing" values

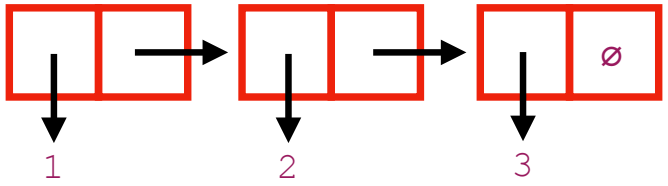
```
(cons "hello" 4)
```



Lisp syntax: lists

- E.g., lists in Lisp are just made out of cons cells

```
(cons 1 (cons 2 (cons 3 nil)))
```



- Lisp has a shorthand for this:

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

“Recursive Functions […]” (McCarthy)

Lisp	C
`car`	`head`
`cdr`	`tail`
`cons`	`prepend`

Note: **`head`** and **`tail`** have a different meaning than the ones you learned in CS136.

Lisp syntax: `car` and `cdr`

- Access the first element of a cons cell with `car`

  `(car (cons 1 2))` ⇝ `1`
- Access the second element with `cdr`

  `(cdr (cons 1 2))` ⇝ `2`
- What's the value of the following expression?

  `(car '(1 2 3))`
- What about this?

  `(cdr '(1 2 3))`

Historical note: `car` and `cdr`

- **`car`** stands for “**C**ontents of the **A**ddress **R**egister”
- **`cdr`** stands for “**C**ontents of the **D**ecrement **R**egister”

These were **instructions** on the **IBM 704**.

Activity: fizzbuzz

Write a program that prints the numbers from **`1`** to **`100`**. But for multiples of three print **`fizz`** instead of the number and for the multiples of five print **`buzz`**.

Helpful bits:

`(and` p_1 … p_n`)`

`T` and `nil`

`(eq` p_1 p_2`)`

`(mod` n_1 n_2`)`

sequence

Recap & Next Class

Today:

LISP

Next class:

Higher-order functions