



EEL5840: Elements of Machine Intelligence

Rocko

## Announcements

- Announcements:
  - > None
- Today's Handouts in WWW:
  - > Outline Class 12
  - > Tentative First Exam Date  
Thur. Oct. 8<sup>th</sup> in class
- Web Site
  - > [www.mil.ufl.edu/eel5840](http://www.mil.ufl.edu/eel5840)
  - > Software and Notes
  - > XLISP Documentation

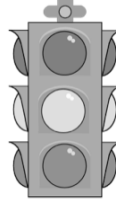
University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Amato Amato

1

EEL5840: Elements of Machine Intelligence

Rocko

## Today's Menu



- Finish LISP LAB 3
  - > Functional Arguments, *funargs*
  - > More on MAPping Functions
  - > I/O Functions
  - > PROG
  - > LOOPing
  - > Association Lists

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Amato Amato

2

EEL5840: Elements of Machine Intelligence

Rocko

## LISP Lab 3

- MAPping Functions  
MAPCAR and related mapping functions allow us to easily transform lists. MAPCAR takes two arguments, the first is a *funarg* and the second a list. The answer is a list of the results of applying the functional argument to each successive car of the 2<sup>nd</sup> argument. Thus, `(mapcar #'f '(x1 x2 x3)) ==> (f(x1) f(x2) f(x3))`  
*template:* `(mapcar #'<fname> <a list of SEXes for fname>)`  
We can define mapcar as follows:
 

```
(defun mymapcar (f lis) (cond
  ( (null lis) nil )
  ( t (cons (funcall f (car lis))
             (mymapcar f (cdr lis))))))
```

`> (mymapcar #'oddp '(1 2 3)) ==> (T NIL T)`

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Amato Amato

3

EEL5840: Elements of Machine Intelligence

Rocko

## LISP Lab 3

- MAPping Functions  
Use a mapping function to transpose an N×N matrix represented as N lists of N integers each, e.g.,
  - Lisp→`(setf mat '((1 2 3) (4 5 6) (7 8 9)))`  
`(1 2 3) (4 5 6) (7 8 9)`
  - Lisp→`(transpose mat)`  
`(1 4 7) (2 5 8) (3 6 9)`
  - Lisp→`(transpose '( (1 2) (3 4) (5 6) ))`  
`(1 3 5) (2 4 6)`
- > `(mapcar #'car mat)`  
`(1 4 7)`
- > `(mapcar #'cdr mat)`  
`((2 3) (5 6) (8 9))`
- Lisp→`(defun transp (mat) (if (null (car mat)) nil (cons (mapcar #'car mat) (transp (mapcar #'cdr mat)))))`
- Lisp→`(transp mat)`  
`((1 4 7) (2 5 8) (3 6 9))`

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Amato Amato

4

EEL5840: Elements of Machine Intelligence

Rocko

### LISP Lab 3

- LAMBDA Expressions allow us to define anonymous procedures**  
The lambda expression is the actual mechanism that the LISP interpreter uses to "evaluate" dummy arguments in a procedure. This is what XLISP returns when you use FUNCTION-LAMBDA-EXPRESSION. Suppose we want to put quotes around all the elements of an input list.

```
(defun put-quotes(lis) (mapcar #'qhelp lis))
(defun qhelp(sex) (list (quote quote) sex))
> (put-quotes '(this is a test))
((QUOTE THIS) (QUOTE IS) (QUOTE A) (QUOTE TEST))
```

But if you begin to run out of names for your "helper" functions (especially if they are only used once in a program) a more elegant solution is given by:

```
(defun put-quotes(lis)
  (mapcar #'(lambda(sex) (list (quote quote) sex))) lis ))
```

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Antonio Arino

5

EEL5840: Elements of Machine Intelligence

Rocko

### LISP Lab 3

- Lambda expressions are the actual execution mechanism for user-defined functions in XLISP. For example, (add1 2) or typing ((lambda (x) (+ 1 x)) 2) yield the same result, a 3 is returned.

```
> (defun add1 (x) (+ 1 x))
ADD1
> (add1 2)
3
(add1 (+ 1 2))
4
>((lambda (x) (+ 1 x)) 2)
3
((lambda (x) (+ 1 x)) (+ 1 2))
4
((lambda (x y) (+ x y)) 2 3)
5
```

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Antonio Arino

6

EEL5840: Elements of Machine Intelligence

Rocko

### LISP Lab 3

- Substop replaces a sex in a list at the top-level. Let's try to write substop using a mapping function.

```
(defun replacesex (sex3)
  (if (equal sex1 sex3) sex2 sex3)) /* sex1,2 are globals! */
/* The following code will not run. Why? */
(defun substop (sex1 sex2 lis) (mapcar #'replacesex lis))
> (substop 'coke 'pepsi '(coke is it))
Error: The variable SEX1 is unbound.
Happened in: #<Closure-REPLACESSEX: #794b20>
```

A lambda expression lets us fix this nicely

```
> (defun substop (sex1 sex2 lis) (mapcar
  #'(lambda (sex3) (if (equal sex1 sex3) sex2 sex3)) lis))
> (substop 'coke 'pepsi '(coke is it))
(PEPSI IS IT)
```

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Antonio Arino

7

EEL5840: Elements of Machine Intelligence

Rocko

### LISP Lab 3

- User-Defined MAPing Functions**  
Using our definition of MAPCAR we can define MAPPENDCAR, which applies f(.) to successive cdrs as follows:

```
(defun mappendcar (f lis) (cond
  ( (null lis) nil )
  ( t (append (funcall f (car lis))
    (mappendcar f (cdr lis))) ) ) )
```

```
(defun inter (s1 s2) (mappendcar #'ihelp s1))
(defun ihelp (s1el) (cond
  ( (member s1el s2) (list s1el))
  ( t nil ) ) )
```

*But this will not work either. Why?*

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Antonio Arino

8

EEL5840: Elements of Machine Intelligence

Rocko

### LISP Lab 3

```
> (inter '(a b) '(b c))
Error: The variable S2 is unbound.
Happened in: #<Closure-IHELP: #76dc8c
```

The reason is obvious(?), s2 is only defined inside INTER but is undefined inside ihelp. We MUST use lambda:

```
> (defun inter (s1 s2) (mappendcar #'(lambda (s1el) (cond
      ((member s1el s2) (list s1el))
      ( t nil)))) s1))

INTER
> (inter '(a b) '(b c))
(B)
```

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Aravamudan

EEL5840: Elements of Machine Intelligence

Rocko

### LISP Lab 4

I/O Functions in LISP

The following represents a sampling of the available I/O functions in LISP. For a complete list see the XLISP Reference Manual. If the file pointers outf or inf are omitted, the system defaults to the terminal.

(setf outf (open "fname" :direction :output))  
Opens a disk file for output. Sets Outf i.e., a file pointer for further use.

(print Sex outf) Returns the value of the Sex and sends a copy to outf.

(dribble "fname") Echoes all CRT I/O to the disk file

(prin1 Sex outf) Same as print except for the trailing cr/lf

(terpri outf) Issues a cr/lf

(read inf) Inputs the next Sex from inf (or CRT if inf omitted)

(read-line inf) Read 1 line from inf

(read-char inf) Read 1 character from inf

(close outf) Close file outf (or inf)

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Aravamudan

EEL5840: Elements of Machine Intelligence

Rocko

### LISP Lab 4

PROG

The PROG construct allows for an indefinite number of forms to be evaluated with local variables. In modern LISP it has been replaced by LET and LOOP constructs, but it is still useful.

(prog (var1 var2 ... varn) <forms>)

1. The number of forms in prog is indeterminate.
2. Prog initializes all its local variables to nil (var1 var2 ... varn).
3. <Forms> are evaluated in order except:
  - a. Atomic forms are not evaluated. They are treated as labels.
  - b. (GO label) means continue from the form that follows label.
  - c. (return expr) exits prog evaluation & returns (eval expr).
4. If prog runs out of forms, it returns nil.
5. If a cond runs out of stuff inside a prog it is treated as a NOOP.

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Aravamudan

EEL5840: Elements of Machine Intelligence

Rocko

### LISP Lab 4

Example:

```
(defun fact (n) (prog (i j)
  (if (< n 0) (return nil) )
  (setf i 1 j 1)
  ( if (< n 2) (return j) )
  LOOP
  (setf i (+ i 1) j (* j i) )
  (if (= i n) (return j))
  (go LOOP)
))
```

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Aravamudan

EEL5840: Elements of Machine Intelligence  
**LISP Lab 4**  
 Rocko

**LOOPING**  
 Common LISP includes most(all) the most popular looping constructs

*template:* (dotimes (<count atm> <upperbound form> <result form>) <body>)

```
> (defun power (m n)
  (let ((result 1))
    (dotimes (count n result) (setf result (* m result))) ) )
```

Also

*template:* (dolist (<element> <list form> <result form>) <body>)

```
> (let ((n 0))(dolist (grade '(john a) (mary b) (james c) (dick a)) (list n 'non-a-grades)) (if (equal (cadr grade) 'a) (print (list 'a (car grade)))) (setf n (+ 1 n))) ) )
```

\* Prints all the A grades in the input list & counts the non-A grades \*

**EVALUATIONS**

```
(eval sex)          hands sex to the interpreter
if (setf x '(+ 1 2)) then          (eval x) returns 3
```

University of Florida  
 EEL 5840 - Class #12 - Fall 2009  
 © Dr. A. Aravamudan

13

EEL5840: Elements of Machine Intelligence  
**LISP Lab 4**  
 Rocko

**PROPERTY LISTS**  
 The ability to associate with an atom in LISP a lisp form (the *property value*) under a given atomic designation (the *property name*).

*template:* (putprop symb pvalue pname) {Requires 3 arguments}

symb - must be atomic, i.e., the atom to receive the property.  
 pvalue - the property value, which can be any sex.  
 pname - must also be atomic, i.e., the property name.

```
> (putprop 'cain 'adam 'father)
ADAM
> (putprop 'eve '(cain abel) 'mother-of)
(CAIN ABEL)
> (setf 'eve 'first-woman)
FIRST-WOMAN
> eve
FIRST-WOMAN
```

University of Florida  
 EEL 5840 - Class #12 - Fall 2009  
 © Dr. A. Aravamudan

14

EEL5840: Elements of Machine Intelligence  
**LISP Lab 4**  
 Rocko

- Putprop is not always included in some LISP dialects. If not use:

```
> (setf (get 'cain 'father) 'adam)
> (setf (get 'eve 'mother-of) '(cain abel))
> (setf (get 'eve 'grandmother) '(tubalcain jubal))
```

```
(defun putprop (atm pvalue pname) (setf (get atm pname) pvalue))
```

In memory, property lists are stored as association lists or dotted pairs associated with the atom, e.g.,

```
(eve (mother (cain abel)) (grandmother (tubalcain jubal)))
```

University of Florida  
 EEL 5840 - Class #12 - Fall 2009  
 © Dr. A. Aravamudan

15

EEL5840: Elements of Machine Intelligence  
**LISP Lab 4**  
 Rocko

- To retrieve properties use the function (get atm pname). It returns the property value if one has been defined or nil. Therefore, the system cannot distinguish between undefined properties and those whose value has been set to nil. A good rule to follow is to always set pvalue to something non-nil

```
> (get 'cain 'father)
ADAM
> (get 'eve 'mother-of)
(CAIN ABEL)
> (get 'cain 'grandfather)
NIL
```

- To remove a property use (remprop symb pname). Returns nil.

```
> (remprop 'cain 'father)
NIL
> (get 'cain 'father)
NIL
```

University of Florida  
 EEL 5840 - Class #12 - Fall 2009  
 © Dr. A. Aravamudan

16

EEL5840: Elements of Machine Intelligence

Rocko

### LISP Lab 4

- Another way to view this is as an associated list of pairs (a-list for short). Associated pairs are lists of sublists where the first element of each sublist is a "key or indicator" and the cdr of the sublist is the value. For example:  

```
> (setq eve '( (mother (cain abel)) (grandmother (tubalcain jubal)) ))  
((MOTHER (CAIN ABEL)) (GRANDMOTHER (TUBALCAIN JUBAL)))  
> eve  
((MOTHER (CAIN ABEL)) (GRANDMOTHER (TUBALCAIN JUBAL)))
```

The function (assoc key a-list) returns the corresponding pair that has a key value matching argument.  

```
> (assoc 'mother 'eve)  
(MOTHER (CAIN ABEL))
```

```
(defun myassoc (key alist) (cond  
  ((null alist) nil)  
  ((equal key (caar alist)) (car alist))  
  (t (myassoc key (cdr alist)))))
```

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Aravamudan

17

EEL5840: Elements of Machine Intelligence

Rocko

# *See LISP Notes 3 The End!*

University of Florida  
EEL 5840 - Class #12 - Fall 2009  
© Dr. A. Aravamudan

18