Intro to LISP

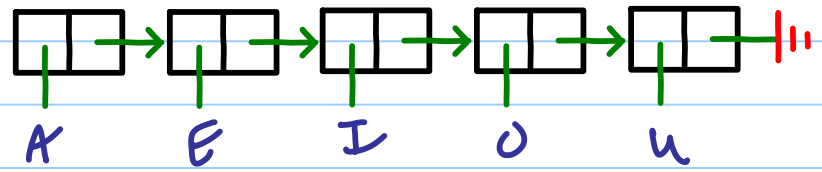
Note Title

LISP — LIST Processing

Lots of Idiotic Stupid Parentheses

Simple representation + manipulation of lists

vowels :



(setf vowels '(A E I O U))

(quote (A E I O U))

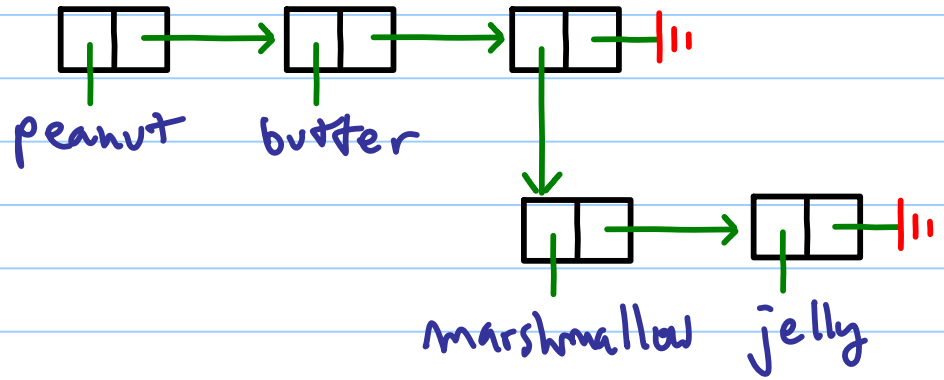
Primitive elements : atoms and lists

atoms — peanut butter jelly "a string" 58

lists — (atoms +/or lists)

(peanut butter jelly)

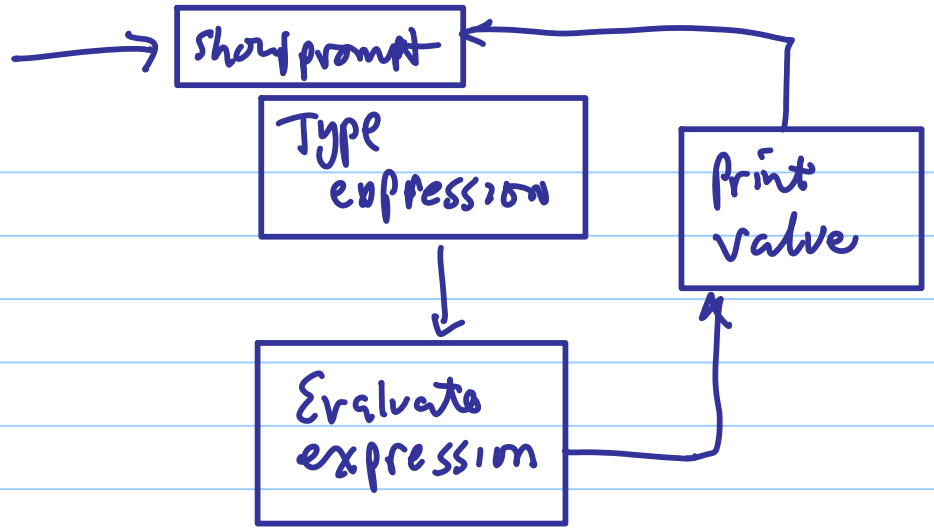
(peanut butter (marshmallow jelly))

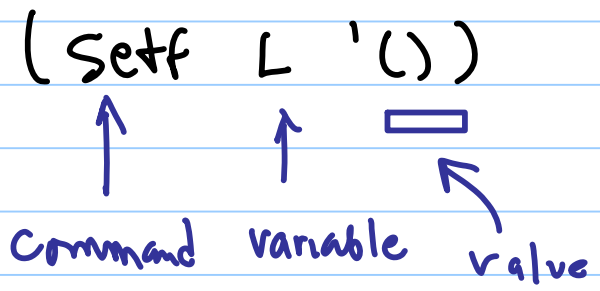


Recommended Reading:
The Little LISPer

NOTE: A list is either NULL or has elements

null list : ()
NIL





Imperative language	Lisp
$f(x)$	(f x)

means apply function f to argument x

$g(x, y, z)$	(g x y z)

In a list, first element is expected to be a function which uses remaining elements as arguments —

||

If the list is data instead of a function call

suppress evaluation with ' (quote)

(setf sandwich '(peanut butter (jelly marshmallow)))

NOTE: In CLISP, if an error is made, you enter a BREAK submenu which can be useful. TO escape, type quit)

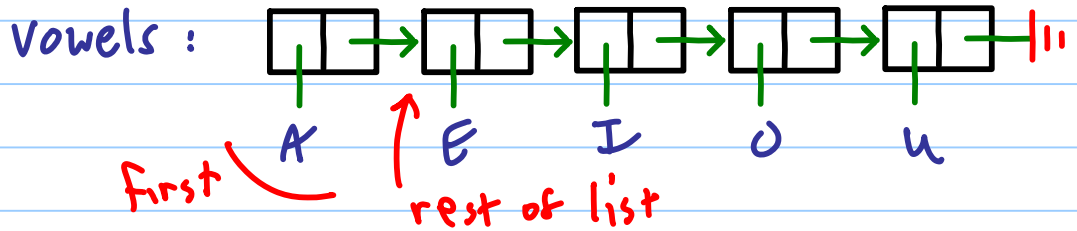
To extract information from a list:

① NOTE: a list contains a first element and a rest of the list, which is itself a list.

The null list does not have a first element —

⊙ To get the first element of L (car L) — (first L)

The rest of L (cdr L) — (rest L)



```
> (car vowels)
A
> (cdr vowels)
(E I O U)
> (car (cdr vowels))
E
> (cadr vowels)
E
```

caddr
cdar
cdadr
⋮
etc.

Defining functions in LISP

```
(defun name (arg list)
  expression
)
```

A problem to solve (someday) in LISP:

A farmer is taking a fox, goose, and bag of corn to market and must cross a river. The river has a boat that can hold the farmer and one item, so he must make multiple crossings while leaving some items unattended. If the fox gets a chance, it will eat the goose; likewise the goose will eat the corn. What's a poor farmer to do?

Representation : (left-bank list right-bank list)

Initial State : ((fox goose corn boat) ())

↑ indicates "farmer + boat"

Tasks: define functions

(leftBank state) : (fox goose corn boat)

(rightBank state) : NIL

Another state: ((fox corn)(goose boat))

(leftBank state) : (fox corn)

(rightBank state) : (goose Boat)

To quit CLISP, type (quit)

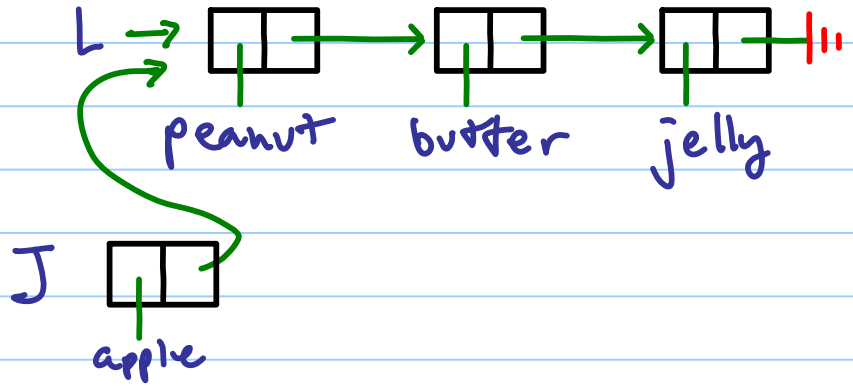
Cons

Note car + cdr can pull parts from lists
cons makes lists

given a list L
and an item X (either an atom or list)

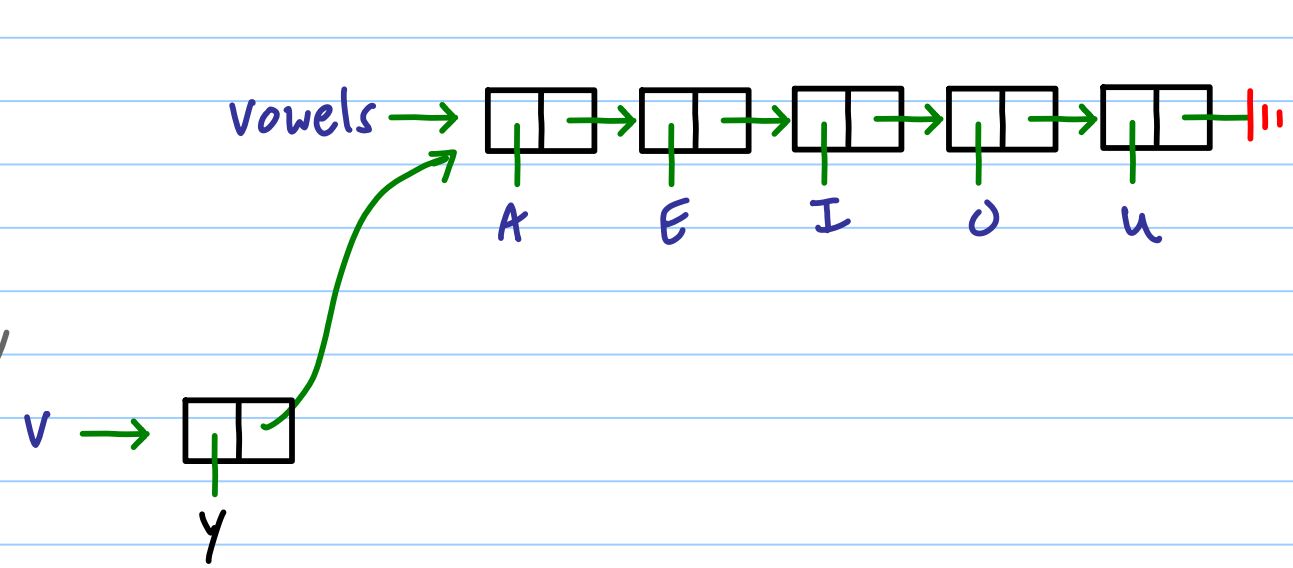
(cons X L) returns a new list (L is unaffected)
with X as first element
L as rest

```
> (setf L '(peanut butter jelly))
> (setf J (cons 'apple L))
(apple peanut butter jelly)
```



cons uses shallow copying — does not create a new copy of a list argument

(setf v (cons 'y vowels))



(Y A E I O U)

The Horrible Truth about Equality:

eq compares two atoms, or whether two pointers point to the same location.

equal compares whether two structures have identical form and values.

```
(setf x '(a (b c) 1 2 3))
(setf y (car (cdr x)))
(setf z (cdr x))
(setf w (car z))
(eq y w) ‘returns T
(eq y ‘(b c)) ‘returns NIL
(equal y ‘(b c)) ‘returns T
```