

Quick intro to Lisp

Not the prettiest, but hey it's functional

Morgan Goose

April 2010

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

This presentation

Want the source? it's all wrapped up for you here:

http://morgangoose.com/p/lisp_intro.tar.gz

Whats is look like

Not as horrible as you may have heard

```
(defun power (x n)
  (cond
    ((= n 0) 1)
    ((= n 1) x)
    (T (* x (power x (- n 1))))))

(assert (eq 256 (power 2 7)))
```

This being a function to raise a number to a given power, and then assert that this function works correctly with a simple test.

Basics

Everything in the same scope is contained in their own (). Functions are the start of the scope, and anything else before the last paren is an argument.

so to add up some numbers:

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

Is the same as this bit of python

```
1 + 2 + 3 + 4
```

Running things

I use *clisp* just because it's what showed up under my package manager when I search for lisp. It's simple enough to use. Just typing *clisp* will drop you into an interactive lisp prompt. If you have a lisp source file *clisp -f file.lisp* will run the program.

```
$ clisp
...
[1]>
```

Control-d will exit so will (quit)

From lists

cons, *car*, *cdr* The most basic lisp list manipulation commands.

```
[1]> (cons 1 (cons 2 (cons 3 nil)))
(1 2 3)

[2]> (car (cons 1 (cons 2 (cons 3 nil))))
1

[3]> (cdr (cons 1 (cons 2 (cons 3 nil))))
(2 3)
```

Short hand for above being

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

Functions

You can make functions pretty simply. Say you want to return a specific list of numbers

```
(defun have_a_string ()
  '(2 3))
```

Or say you want to return $x + x$

```
[1]> (defun double (x)
  (+ x x))

[2]> (double 10)
20
```

Conditionals and recursion

```
(defun factorial (n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1)))))
```

This block will take a parameter n if it's less than or equal to 1 it'll return the factorial base case ($0! == 1$), other wise it'll recurse down until it does and then multiply each result back up until it reaches the first call.

Fun stuff.

I figure at this point my 5 min are up

Here are some good places to look for more lisp info:

http://en.wikibooks.org/wiki/Common_Lisp/First_steps/Beginner_tutorial

<http://www.cs.sfu.ca/CC/310/pwfong/Lisp/1/tutorial1.html>

http://en.wikibooks.org/wiki/Common_Lisp

http://en.wikipedia.org/wiki/Common_Lisp

HN had a long thread:

<http://news.ycombinator.com/item?id=280118>

Also for you java mongers:

<http://clojure.org/>