BRIEF HISTORY

• Developed in 1958 as a functional language
  • Common LISP
    • Supports procedural, functional, and object oriented programming paradigms
  • Scheme – smaller, less built in features, no macros

• Pioneer of many ideas in CS
  • Garbage collection
  • Dynamic Typing

• The name derives from “LISt Proccessing”
  • Linked lists, main data structure
S-EXPRESSIONS

- **Lists**
  - Surrounded by parenthesis
  - Made up of lists and atoms
  - LISP lists start with a Symbol

- **Atoms**
  - Everything that isn’t a list AKA Symbols
    - Variables
    - Functions names
    - T and nil – the truth values in LISP
    - Keyword
    - Symbols
    - Numbers
    - Strings
    - Chars

(\(+\) (\(*\) 6 8) 5)
BASIC LAYOUT OF FUNCTIONS

• \( (\text{function-name parameters}) \)
  • \(+ 3 4 5 6\)
    • \(+\) is the function name
    • \(3 4 5 6\) are the parameters for the function

• \( (\text{defvar *owl* 1}) \)
  • \(\text{defvar}\) – function name
  • \(*\text{owl*}\) - symbol
  • \(1\) – the value to set to the symbol
LISTS

- **Cons** – Node
- **Car** – Node’s value  
  Synonym: first
- **Cdr** – Node’s next pointer  
  Synonym: rest
Lisp is not a case sensitive language

- `owl` == `Owl` == `OWL`

- CamelCase is not used much in list
- `-` is used most frequently to separate words in lisp

**Special Characters in Lisp**

- `(`  `)`  `"`  `'`  `\` - you can still include them in names

**Valid names**

- `owls-can-fly` → `OWLS-CAN-FLY` ; a regular name
- `Owls\ can\(fly\)` → `| OWLS CAN(FLY) |` ; escaping characters
- `O|wls can |fly` → `| Owls can FLY |` ; using the `||` to escape list of chars
- `1guffo` → `1GUFFO` ; Numbers at the beginning of a name
- `14.8x3` → `| 14.8X3 |` ; A name that looks like a number
- `8+9` → `| 8+9 |` ; Syntax with meaning in other languages

**Invalid names**

- A name of all periods
- Anything that can actually evaluate to a number

**Global Variables** `*OWL*`

**Constants** `+OWL+`
NUMBERS

• The syntax matters in identifying the type of a number
• Integers – can be arbitrarily large
  • -313  Negative
  • 313   Positive
  • 626/2 Another representation
  • #b0101 Binary 5
  • #x1AF  Hex 431

• Ratios
  • 1/2 The ratio one over two
  • (+ 1/4 1/8) → 3/8

• Floating Point
  • 3.0  Single precision floating point
  • 3.0e2 Single precision 300
  • 3.0d0 double point precision

• Complex numbers
  • #c(real imaginary) → both the real and imaginary are ratios or floating point
FUNCTIONS FOR NUMBERS

• + - * / 
• floor - rounds to negative infinity 
• mod – is % in other language 
• Conditionals 
  • ( = number number) – Equivalent to == 
  • (/= number number) – Equivalent to != 
  • (< number number) – Equivalent to less than 
  • (<= number number) – Equivalent to less than or equal to 
• ++, --, +-, -= \( \rightarrow (incf \ x) \ (decf \ x) \ (incf \ x \ 10) \ (decf \ x \ 10) \) 
• and and or are there own functions
ASSIGNMENT

• (defvar owl 10)
• (defparameter owl 10)
• (let ((owl 5) (fly "I can Fly")) ...)  
• Reassigning  
  • (setf owl 5)
SCOPE

- Functions create their own scope

```lisp
(defparameter owl 1) ; global owl

(defun test (owl)
  (let ((owl 10) (fly 20))
    (format t "~d ~\%" owl))
  (format t "~d ~\%" owl))
  (format t "~d ~\%" owl))
  (test 6) ; Test the function

; function owl
; let owl
; print 10
; print 6
; print 1
```
FUNCTIONS

; Binary Search
(defun binSearch (value array min max)
  ;if max is less then min return nil
  (if (< max min)
      nil
    ;otherwise find the midpoint
    (let ((midpoint (floor (/ (+ max min) 2)))))
      ; use a cond statement to set up 3 options
      (cond
        ((< (nth midpoint array) value)
         (binSearch value array (+ 1 midpoint) max))
        ((> (nth midpoint array) value)
         (binSearch value array min (- midpoint 1)))
        (T midpoint))))))
FUNCTION AS ARGUMENTS

; defines a function that takes a function and a number as a parameter x and returns the value of calling the function with x
(defun callfun(f x)
    (* x (funcall f x)))

; A call that will cube 6
(callfun #'(lambda (x) (* x x)) 6)
PARAMETERS

&optional

&rest

&key

(defun foo ( o w &optional (1 6) s)  
  (list o w 1 s))

(foo 6 2)  --->  (6 2 6 nil)
(foo 6 2 7)  --->  (6 2 7 nil)
(foo 6 2 6 1)  --->  (6 2 6 1)

(+ 1 2) (+ 1 2 3) (+ 1 2 3 4) ...
(* 1 2) (* 1 2 3) (* 1 2 3 4) ...

; Basic format for &rest
(defun + (&rest numbers) ...)

(defun foo ( &key o w 1 (s 6))  
  (list o w 1 s))

(foo)  --->  (nil nil nil 6)
(foo :o 5)  --->  (5 nil nil 6)
(foo :w 6 :o 5)  --->  (5 6 nil 6)
MACROS

• Are used to add functionality and syntax to the language.
• A function to add functionality directly to the language
• defvar is an example of a macro it returns actual lisp code that can be called and run.
• Its pretty cool
Last night I drifted off while reading a Lisp book.

Huh?

Suddenly, I was bathed in a suffusion of blue.

At once, just like they said, I felt a great enlightenment. I saw the naked structure of Lisp code unfold before me.

My God, it's full of car's.

The patterns and meta-patterns danced. Syntax faded, and I swam in the purity of quantified conception. Of ideas manifest.

Truly, this was the language from which the gods wrought the universe.

No, it's not.

It's not?

I mean, ostensibly, yes. Honestly, we hacked most of it together with Perl.