

- Borrows extensively from Geoff Gordon's "Lisp Hints" (1993)

Numbers

- Integers
- Reals
- Rationals: $2/3$, $-17/23$

Standard Arithmetic Functions

- $+$, $-$, $*$, $/$
- floor, ceiling, mod
- sin, cos, tan, sqrt
- Prefix notation - think "application of a function to its arguments"
- Return a number according to type contagion:
 - integer and rational \rightarrow rational
 - rational and real \rightarrow real

Some Examples

```
> (+ 3 3/4)
15/4
> (/ 10.0 5)
2.0
> (/ 10 5 5 1)
2/5 ; +-* / accept
      ; multiple arguments
```

Symbols

- A symbol is just a string of characters
- As long as you stick to letters, digits, and hyphens, you'll be safe, but
- Other characters are allowed

Examples

- T1
- Trouble-234
- a
- b
- Lisp is not case sensitive: Trouble-234 is like TROUBLE-234 is like trouble-234 etc.

Special Symbols

- Two special symbols, `t` and `nil`
 - The value of `t` is always `t`
 - The value of `nil` is always `nil`
- `nil` is taken to mean false; anything else is taken to mean true.
- `nil` is also used in other contexts.

Conses

- A `cons`
 - is just a two-field record
 - CAR = “contents of address register”
 - CDR = “contents of decrement register”
 - Or a binary tree

Examples

```
> (cons 4 5)
(4 . 5)
> (cons (cons 4 5) 6)
((4 . 5) . 6)
> (car (cons 4 5))
4
> (cdr (cons 4 5))
5
```

- In the examples above,
 - 4 and 5 are **atoms**
 - The conses are **dotted lists**

Lists

- Built out of conses
- `car` is the head of the list
- `cdr` is the rest - either another cons or `nil`

Examples

```
> (list 4 5 6)
(4 5 6)
> (cons 4 nil)
(4)
> (cons 4 (cons 5 6))
(4 5 . 6)
> (cons 4 (cons 5 (cons 6 nil)))
(4 5 6)
```

- The last example is equivalent to the call
`(list 4 5 6)`

“nil” as an empty list

- In addition to meaning false, `nil` also represents an empty list.
- `car` and `cdr` of `nil` defined to be `nil`.
- Equivalent to `' ()`

A special form: quote

- Used to suppress evaluation
 - `(cons 'a 'b) -> (A . B)`
 - `(cons 'a '(b c d)) -> (A B C D)`
- What would `(cons a (b c d))` mean?
- Can also say
 - `(cons (quote a) (quote b))`
 - `(cons (quote a) (quote (b c d)))`

Defining Simple Functions

- $f(x, y) = x^2 + y^2$

```
>(defun sum-of-squares(x y) (+ (* x x) (* y y)))
SUM-OF-SQUARES
>(sum-of-squares 2 3)
13
```
- Line breaks are ok! (a good thing!)

```
(defun sum-of-squares (x y)
  (+ (* x x)
     (* y y)))
```

Optional Arguments

- Add `&rest` parameter to end of argument list
- Lisp collects args not accounted for into a list
- Binds `&rest` parameter to that list
- Can also specify `&optional` params

Example

```
>(defun give-me-the-rest (x &rest y) y)
GIVE-ME-THE-REST
>(give-me-the-rest 3)
NIL
>(give-me-the-rest 4 5 6)
(5 6)
```

Forms and the top-level loop

- The things you type into the Lisp interpreter are called **forms**
- Read-eval-print loop:
 - If form = atom(integer, string, symbol - i.e., not a list), eval immediately.
 - if form = list, the car of the list is a function, which is applied to its args.
- **Special forms**
 - Look like function calls but aren't
 - Include `defun` and `quote` (also `if`, `do`, `setq`, and others)

Functions that get at list elements

- `car` returns the first element of a list
- `cdr` returns the rest
- Can use the functions `first` and `rest` instead of `car` and `cdr`

Examples

```
(car '(a b c d)) -> A
(cdr '(a b c d)) -> (B C D)
(car '((a b c) d)) -> (A B C)
(cdr '((a b c) d)) -> (D)
```

- Note: car and cdr of nil, '() and '(nil) are nil; while nil and '() are equal, '(nil) is not the same as either.

caar, caaar, cddr, cadr, caddr...

- Often want to apply car or cdr to the result of an application of car or cdr
- Shorthand function names
 - cadr returns the car of the cdr of the arg
 - caar returns the car of the car of the arg
 - cdar returns the cdr of the car of the arg
- Can combine up to 4 cars/cdrs in this way

More ways to get at list elements

- If x is a list,
- (nth 4 x) returns the element at index 4 (0-indexing)
- (fifth x) returns the same value - i.e., the fifth element in the list
- Can use first, second, ..., tenth

Building lists

- cons
 - list
 - append - concatenates two lists
- ```
> (append '(1 2 3) '(4 5 6))
(1 2 3 4 5 6)
> (append '(1 2) '((3 4) 5 6))
(1 2 (3 4) 5 6)
```

## What are lists good for?

- Flexible data structure
- (1 8 6 ...) can represent the positions of queens in 8-queens problem
- ((1 2 3) (4 5 6) (7 8 9)) can represent the goal state in the 8-puzzle
- (The dog chased the cat) can represent a specific sentence

## Some useful list functions

- length, list-length (better practice to get used to using list-length)
  - reverse
- ```
> (length '(a b (c d)))
3
> (reverse '(a b (c d)))
((C D) B A)
```
- NB: Be careful when applying reverse to a final cons with a non-null cdr.

Conditionals

- Several ways to write conditionals in Lisp
- Cond macro:

```
(cond (test-1 consequent-1.1 consequent 1.2 ...)  
      (test-2 consequent-2.1 consequent 2.2 ...)  
      (test-3 consequent-3.1 consequent 3.2 ...)  
      ...)
```

How cond works

- The first clause whose test evaluates to non-nil is selected. All others are ignored.
- Remember that tests return either nil or a non-nil value (which is taken to mean true)
- Value of cond form is either
 - the result of the last consequent executed or
 - the result of the selected test or
 - nil.

Predicates

- Functions that return either nil or a non-value to mean “true”
- Can be viewed as boolean functions
- Examples of some useful predicates
 - `null` - whether its single argument is a list
 - `listp` - whether its argument is a list or not
 - `atom` - whether its argument is an atom or not

The many faces of equality

- Different functions test for different types of equality
 - Numeric equality is denoted by `=`
 - Two symbols are **eq** if and only if they are identical
 - Two copies of the same list are not `eq`, but they are **equal**

Examples

```
> (eq 'a 'a)  
T  
> (eq 'a 'b)  
NIL  
> (= 3 4)  
NIL  
> (eq '(a b c) '(a b c))  
NIL  
> (equal '(a b c) '(a b c))  
T  
> (eql 'a 'a)  
T  
> (eql 3 3)  
T
```

... and more on equality

- `eql` predicate is equivalent to `eq` for symbols and to `=` for numbers
- `equal` predicate is equivalent to `eql` for symbols and numbers; true for conses iff their cars are equal and their cdrs are equal.

member

- Assumes that the test to be performed is eq

```
> (member 'a '(a b c))
(A B C) ; true
> (member '(a b) '((a b) c))
NIL
```
- To test with notion of equal:

```
(member thing-to-find list-to-search
:test 'equal)
```

Putting it all together

- We now have the basic tools for writing more interesting functions:
 - Data: atoms and lists
 - Functions
 - Ability to express conditionals
 - Predicates