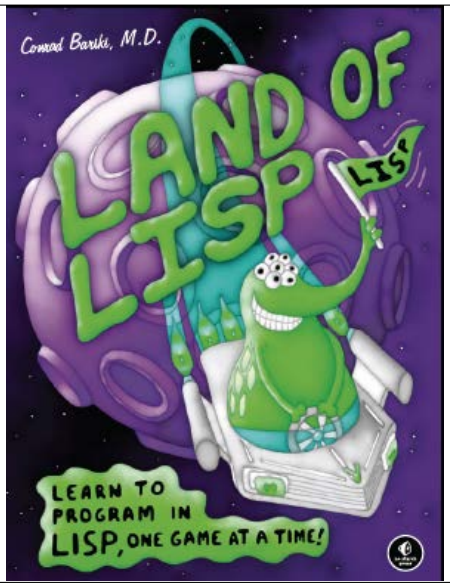
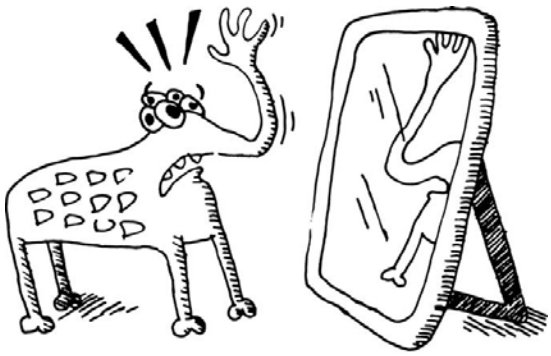


Ch 4



SECTION II:
LISP IS SYMMETRY

Chapter 4
Making Decisions

- True and False
- Control flow - selection and conditional statements
  - `IF`
  - `WHEN, UNLESS`
  - `COND`
- Equality with `EQ, EQL, EQUAL, EQUALP`

True and False

- **False** is represented by both

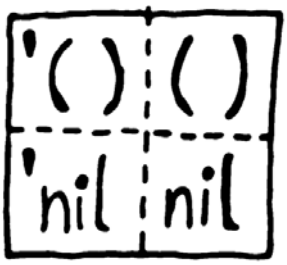
```
nil     ; symbol for the nothing
()      ; symbol for the empty list
```

- **True** is represented by

```
t       ; a special symbol meaning true
```

Anything that isn't nil is considered true!

The Four Forms of NIL



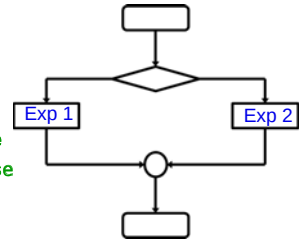
Symmetry: All of these evaluate to `nil`!

IF - Selecting One of Two Paths

- `If` selects between *expression one* if the test evaluates to true, and *expression two* if the test evaluates to false
- Eg, `true` when 1 + 2 = 3
- `False` when 1 + 2 = 4

```
(if
  (= (+ 1 2) 3); test
  'yup  ; evaluate if test true
  'nope); evaluate if test false
```

→ **YUP**




```
(if
  (= (+ 1 2) 4)
  'yup
  'nope)
```

→ **NOPE**

IF is a Special Form

```
(if <test>
        <then-form>
        <else-form> )
```

- Calling a **special form** looks exactly like a function call, however there is a crucial difference, a **special form** does not evaluate its parameters in the normal way!!
- Parameters are **evaluated only when needed**, depending on the specific command's syntax.

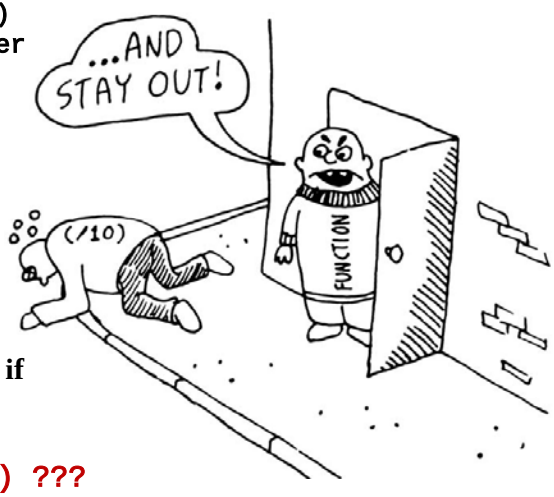
- Example

```
(if (= x 0)     ; test
    0           ; do this if test true
    (/ 10 x))   ; do this if test false
```

- Set x to 0 → returns 0
- Set x to a nonzero number → returns 10/X
- If X is not a number → error
- Only one of the expressions after the `if` is actually evaluated.
- We can do only one thing in each part of an `if` statement (why?)

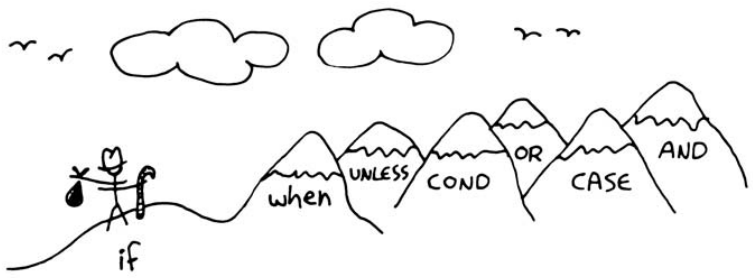

```
> (if (oddp 5)
      'odd-number
      (/ 1 0))

→ODD-NUMBER
```

Doesn't evaluate `(/ 1 0))` **because** `(oddp 5)` **returns true.**



What happens if `oddp` **returns false?**
E.g. `(oddp 6) ???`

Beyond IF -- Other Selection Statements



When and Unless

- With **`when`**, all the enclosed expressions are evaluated when the condition is **true**.
- With **`unless`**, all the enclosed expressions are evaluated when the condition is **false**.
- The trade-off is that ***these commands can't do anything when the condition evaluates in the opposite way***;

  they **`return`** *`nil`* and **`do`** *`nothing`*.

When and Unless Examples

```
> (defvar *number-is-odd* nil) ; create variable
> (when (oddp 5)
        (setf *number-is-odd* t)
        'odd-number)
ODD-NUMBER       ; return value = symbol odd-number
> *number-is-odd*
T     ; check if the value of the variable has changed
> (unless (oddp 4)
          (setf *number-is-odd* nil)
          'even-number)
EVEN-NUMBER      ; return value = symbol even-number
> *number-is-odd*
NIL    ; check if the value of the variable has changed
```

COND
The Multi-way Conditional That Does It All

- **COND** is the most powerful multi-selection statement *in the world!*
- `cond` uses parentheses to separate each test/condition and its corresponding actions
- Tests are always done from the top down
- **All actions** in the **clause after** the **first** successful test are executed.
- Then the `cond` returns the value of the last expression evaluated and then exits - no further tests or clauses are examined.

COND Syntax

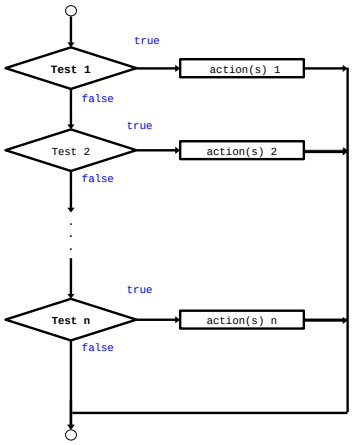
```
(COND
   ( test_1   <expr>* )
   ( test_2   <expr>* )
   ...
   ( test_t   <expr>* )
   {( T <expr>* )}) ;T and ELSE
                    ;both equal true
```

Returns the ***value*** *of the last expression in the first list whose test [predicate] evaluates to non-nil (= true)*

COND is a Special Form

Evaluation of COND

1. Start with the *first clause* (i = 1)
2. Evaluate test i
   - If test is non-nil (TRUE),
     a. Execute all actions in the clause,
     b. Return the value of the last action (result of the `cond` statement) and exit
3. Else increment i, and repeat from step 2 until out of clauses.
   Return `nil` if none of the tests returns `true`



COND Example

```
(defun testx (x)
  (cond
     ((< x 1)          ; first clause test
                       ; rest of 1st clause
        (princ "x is a small number"))

      ((>= x 2)        ; second clause test
                       ; rest of 2nd clause

        (princ "x is a larger number"))
      ( T              ; last clause test
        (princ " X is 1"))
   )

 (testx 5)
 x is a large number              ; prints
 ➔ "x is a large number"          ; returns
```


AND and OR
The Stealth Conditionals

- Use ***shortcut Boolean evaluation***
- `AND` continues evaluating parameters *until one of them is* **false.**

If all are true, the value of the last expression evaluated is returned, else nil is returned.

- `OR` continues evaluating *parameters until one of them is* **true.**

The value of that non-nil expression is returned. If none of the expressions evaluates to non-nil, nil is returned.

AND Example

```
(and *file-modified*
     (ask-user-about-saving)
     (save-file) )  ; do if both 1&2 are true
```

Is equivalent to

```
(if *file-modified*
    (if (ask-user-about-saving)
        (save-file) ))
```

Using Functions That Return More than Just True or False

```
> (member 1 '(3 4 1 5))
(1 5) ; returns the matching item
      ; plus the rest of the list
> (if (member nil '(3 4 nil 5))
    'nil-is-in-the-list
    'nil-is-not-in-the-list)
NIL-IS-IN-THE-LIST
```

It is not necessary to evaluate an expression twice (or save the result in a variable) in order to both test and use the value

Function FUNCTION and #'

```
> (find-if #'oddp
           '(2 4 5 6))
5
> (if (find-if (function oddp)
                '(2 4 5 6))
      'there-is-an-odd-number
      'there-is-no-odd-number)
THERE-IS-AN-ODD-NUMBER
```

`#'` is equivalent to `(function …)`

in exactly the same way as

`'` is equivalent to `(quote …)`

Comparing Stuff: eq, equal, and more

eq, eql, equal, and equalp

Most strict test is **eq**

- **eq** for symbols

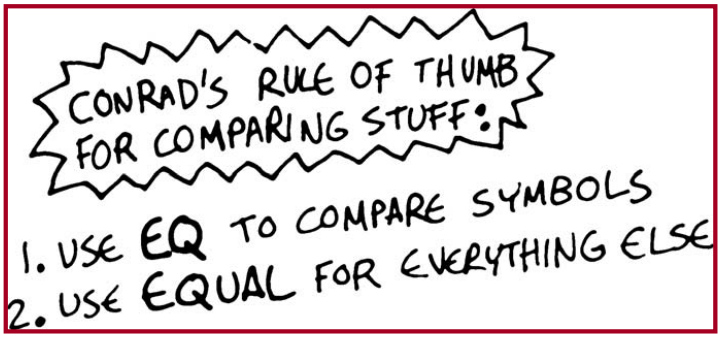
Least strict test is **equalp**

Given the same parameters:

- If **eq** returns **t**, all other tests (**eql, equal, equalp**) will also return **t**
- If **eql** returns **t**, **equal** and **equalp** will also return **t**
- If **equal** returns **t**, **equalp** will return **t**

eq is Strict!

```
;;comparing symbols
> (eq 'apple 'apple)
T
;;comparing lists
> (eq (list 1 2 3) (list 1 2 3))
nil
;;Identical lists created in different ways
> (eq '(1 2 3) (cons 1 (cons 2 (cons 3 ()))))
nil
;;comparing integers
> (eq 5 5)
T
;;comparing floating point numbers to integers
> (eq 2.0 2)
nil
;;comparing strings
> (eq "foo" "foo")
T
;;comparing characters
> (eq #\a #\a)
T
```

EQ Example

```
> (defparameter *fruit* 'apple)
*FRUIT*

> (cond ((eq *fruit* 'apple)
         'its-an-apple)
        ((eq *fruit* 'orange)
         'its-an-orange))
ITS-AN-APPLE
```

The `eql` command is similar to `eq`, but unlike `eq`, it also returns true when comparing ***real numbers***

More Examples

```
;;comparing symbols, integers or characters
> (eq 'foo 'foo)
T
> (eq #\a #\a)
T
> (eq 9 9)
T
;;comparing floating point numbers
> (eq 3.4 3.4)
nil
> (eql 3.4 3.4)
T
;;comparing strings with identical case
> (eql "Bob Smith" "Bob smith")
nil
> (equal "Bob Smith" "Bob smith")
T
;;comparing strings with different case
> (equal "Bob Smith" "bob smith")
nil
> (equalp "Bob Smith" "bob smith")
T
;;comparing integer and floating point numbers
> (equal 0 0.0)
nil
> (equalp 0 0.0)
T
```


Summary

- `t` and `nil` (True and False)
- Selection and condition control flow
  - `if, when, unless,`
  - `cond`
- Stealth conditionals – `, or`
- Equality with =, `eq, eql, equal, equalp`
- Conrad's rule
- Extra examples in following slides

Eql is True More Than EQ

```
;; comparing symbols
> (eql 'apple 'apple)
T
;; comparing lists
> (eql (list 1 2 3) (list 1 2 3))
nil
;; Identical lists created in different ways still
compare as the same
> (eql '(1 2 3) (cons 1 (cons 2 (cons 3 ()))))
nil
;; comparing integers
> (eql 5 5)
T
;; comparing floating point numbers to integers
> (eql 2.5 2.5)
nil
;; comparing strings
> (eql "foo" "foo")
T
;; comparing characters
> (eql #\a #\a)
T
```


Equal is True More Than Eql

```
;; comparing symbols
> (equal 'apple 'apple)
T
;; comparing lists
> (equal (list 1 2 3) (list 1 2 3))
T
;; Identical lists created in different ways still
;;  compare as the same
> (equal '(1 2 3) (cons 1 (cons 2 (cons 3 ()))))
T
;; comparing integers
> (equal 5 5)
T
;; comparing floating point numbers
> (equal 2.5 2.5)
T
;; comparing strings
> (equal "foo" "foo")
T
;; comparing characters
> (equal #\a #\a)
T
```

Equalp Returns True the Most

```
;; comparing lists
> (equalp (list 1 2 3) (list 1 2 3))
T
;; Identical lists created in different ways still
;; compare as the same
> (equalp '(1 2 3) (cons 1 (cons 2 (cons 3 ()))))
T
;; comparing strings with different case
> (equalp "FOO" "foo")
T
;; comparing characters
> (equalp #\a #\a)
T
```