At the start of this session we examine a basic compare mechanism, at this stage we are trying to write our own version of Lisp's *equal* function. This is our first attempt...

```
(defun cmp (pat lis)
  (cond ((and (null pat) (null lis))
         t
         )
        ((or (null pat) (null lis))
         nil
         )
        ((eq (first pat) (first lis))
         (cmp (rest pat) (rest lis))
         )
        (t
         nil
         )
        ))
```

it works as expected/intended for some cases

```
cg-user(5): (cmp '(a b c) '(a b c))
t
cg-user(6): (cmp '(a b c) '(a b x))
nil
cg-user(7): (cmp nil nil)
t
cg-user(8): (cmp '(a b c) '(a b c d))
nil
cg-user(9): (cmp '((a b)(c d)) '((a x)(c d)))
nil
```

but there are test cases where it does not do what we want, eg...

```
cg-user(10): (cmp '((a b)(c d)) '((a b)(c d)))
nil
```

this is because we have used eq in our definition and eq behaves in a specialised way...

```
cg-user(11): (eq 'a 'a)
t
cg-user(12): (eq 'a 'b)
nil
cg-user(13): (eq '(a b c) '(p q r))
nil
cg-user(14): (eq '(a b c) '(a b c))
nil
```

you may not have expected the last result (above). The eq function checks that its 2 args are the same, not just that they look the same or have the same content. For an alternative

function which test equality less stringently (ie: whether 2 structures have the same structure & content) we can use equal.

```
cg-user(15): (equal '(a b c) '(a b c))
t
```

check the following comparison of eq & equal

```
cg-user(16): (setf a '(cat bat mat))
(cat bat mat)
cg-user(17): (setf b '(cat bat mat))
(cat bat mat)
cg-user(18): (setf c a)
(cat bat mat)
cg-user(19): (equal a b)
t
cg-user(20): (equal a c)
t
cg-user(21): (equal b c)
t
cg-user(22): (eq a c)
t
cg-user(23): (eq a b)
nil
```

so we could replace the eq in our previous definition of cmp with equal but we want to go on developing cmp to handle special symbols like "=" and "?". If we use equal to check equivalence of sublists we will not be able to include these special symbols in those sublists so a better solution is to use cmp to match sublists when we find them, as below.

```
(defun cmp (pat lis)
  (cond ((and (null pat) (null lis))
         t
         )
        ((or (null pat) (null lis))
         nil
         )
        ((eq (first pat) (first lis))
         (cmp (rest pat) (rest lis))
         )
        ((and (listp (first pat))
              (listp (first lis)))
         (and (cmp (first pat) (first lis))
              (cmp (rest pat) (rest lis)))
         )
        (t
         nil
         )
        ))

cg-user(24): (cmp '((a b)(c d)) '((a b)(c d)))
t
cg-user(28): (cmp '((a b)(c d)) '((a spam)(c d)))
nil
```

the use of and in the above example is worth investigation further. The semantics of "and" & "or" in Lisp (and in many other languages) is more specialist than many people think.

To investigate use of and & or I will define a couple of symbols (count & daft) whose use may look slightly strange.

```
(defparameter count 0)

(defun daft (x)
  (setf count (1+ count))
  x)
```

the function "daft" increments count & then returns whatever value it was given for its single arg.

```
cg-user(31): (daft t)
t
cg-user(32): (daft nil)
nil
cg-user(33): (daft 'an-egg-sandwich)
an-egg-sandwich
```

note that after this, count is 3

```
cg-user(34): count
3
```

no remember, in lisp, any value which is not nil is considered as true for testing purposes.

```
cg-user(35): (setf count 0)
0
cg-user(36): (and (daft t)
                  (daft 'banana)
                  (daft (eq 'banana 'mango))
                  (daft 'oooer)
                  )
nil
cg-user(37): count
3

cg-user(38): (setf count 0)
0
cg-user(39): (or  (daft t)
                  (daft 'banana)
                  (daft (eq 'banana 'mango))
                  (daft 'oooer)
                  )
t
cg-user(40): count
1
```

```
cg-user(41): (setf count 0)
0
cg-user(42): (or  (daft 'melon)
                  (daft 'banana)
                  (daft (eq 'banana 'mango))
                  (daft 'oooer)
                  )
melon
cg-user(43): count
1
```

notice that or (above) returned "melon". This is because or returns the first non-nil value it finds.

```
cg-user(44): (or  (daft (= 5 161))
                  (eq 'kipper 'cod)
                  (daft 'melon)
                  (daft 'banana)
                  (daft (eq 'banana 'mango))
                  (daft 'oooer)
                  )
melon
cg-user(45): count
3
```

and returns nil or its last value if they are all non-nil

```
cg-user(46): (and 'bat 'rat 'mat 'sat (= 2 3))
nil
cg-user(47): (and 'bat 'rat 'mat 'sat)
sat
```

the behaviour of and & or allows us to build a conditional form like if, check this out...

```
(defparameter test nil)
(defparameter res1 'kipper)
(defparameter res2 'mango)

cg-user(48): (if test res1 res2)
mango

(defparameter test 'horse)

cg-user(49): (if test res1 res2)
kipper
cg-user(51): test
horse

cg-user(52): (or (and test res1) res2)
kipper

(defparameter test nil)

cg-user(53): (or (and test res1) res2)
mango
```

ok, back to the cmp function & now we will add the clause which takes care of the "=" symbol in the pattern arg. NB: the function would work the same if we placed this new clause in a couple of other different places but we prefer it here because it is a more general case than the later clauses.

```
(defun cmp (pat lis)
  (cond ((and (null pat) (null lis))
         t
         )
        ((or (null pat) (null lis))
         nil
         )
        ((eq '= (first pat))
         (cmp (rest pat) (rest lis))
         )
        ((eq (first pat) (first lis))
         (cmp (rest pat) (rest lis))
         )
        ((and (listp (first pat))
              (listp (first lis)))
         (and (cmp (first pat) (first lis))
              (cmp (rest pat) (rest lis)))
         )
        (t
         nil
         )
        ))


cg-user(54): (cmp '((a b) (c d)) '((a spam) (c d)))
nil
cg-user(55): (cmp '((a b) (c d)) '((a b) (c d)))
t
cg-user(56): (cmp '(= (c d)) '((a b) (c d)))
t
```

note that (eq a b) can be true even when a and b are both sublists

```
cg-user(59): (setf fruit '(mango melon pawpaw))
(mango melon pawpaw)
cg-user(60): (setf x (list 'a 'b fruit 'c))
(a b (mango melon pawpaw) c)
cg-user(61): (setf y (list 'n 'm fruit 'kj))
(n m (mango melon pawpaw) kj)
cg-user(62): (eq (third x) (third y))
t
```