Puzzle

Can you write a non-trivial scheme expression that prints itself?

Symbols and quote

Difference between name and object

Quote allows you to manipulate the symbol rather than the value it is bound to.

quote and sugar ’

The first thing that happens to ' is to be desugared into (quote)

predicates: eq? symbol?

To determine if two numbers are identical, we use the Scheme primitive =. To determine if two symbols are identical, we use the Scheme primitive eq?.

For example,

```
(eq? 'apples 'apples) #t
(eq? apples 'oranges) #f
```

eq? takes constant time, regardless of the length of the symbol name

The behavior of eq? on numerical arguments is **unspecified**:

`(eq? 1 1)` ➔ unspecified

`(let x (+ 1 2) (eq? x x))` ➔ unspecified

Strings vs. symbols

- what's the difference between a string & a symbol?
  - strings can have spaces
  - symbols must be legal names (can't start with a number, can't have spaces, parentheses or some special characters)
  - strings are used & printed inside of double quotes
  - symbols print without quotes
  - symbols can be compared in constant time
- why do we need both?
  - by using symbols we can make things that print out like valid scheme expressions
  - symbols can be compared in constant time

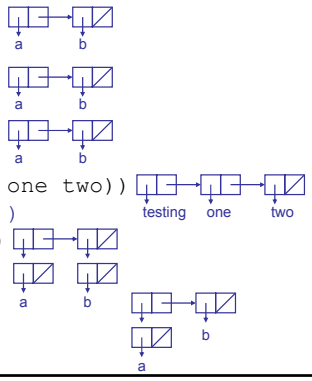
The quiz was too easy

Don't let it go to your head.

We don't want you to have a bad reality check for quiz 2

Give the printed representation and the box-and-pointer diagram for each of the following:

```
>'(a b)
(a b)
>(cons 'a '(b))
(a b)
>(list 'a 'b)
(a b)
>(quote (testing one two))
(testing testing )
>(list '(a) '(b))
((a) (b))
>(cons '(a)'(b))
((a) b)
```



Given the define below, what do the following expressions return?

```
(define a 3)
(define b 4)
(define c (list 5 6))
(define x '(a b))

(list 'a b)     ➔ (a 4)
'(a b)          ➔ (a b)
(cons b x)      ➔ (4 a b)
(list 'b c)     ➔ (b (5 6))
(quote 1 2 3)   ➔ Error, incorrect number of arguments
```

Puzzle

Can you write a non-trivial scheme expression that prints itself?

(Hint: Consider how a cell reproduces. It contains its own blueprint (the DNA strand in the nucleus); reproduction involves (a) copying the blueprint, (b) implementing the blueprint. That is, it uses the blueprint twice.)

```
(define p (list cons +))
(define q '(cons +))
(define r (list 'cons '+))
(define x '(a b))
(define alpha '(a Greek letter))
(define beta '(I don't know))
(car x)          ➔ a
(cdr alpha)      ➔ (Greek letter)
((car p) 3 4)    ➔ (3 . 4)
(cadr p) 3 4)    ➔ 7
((car r) 3 4)    ➔ Error: can't apply a symbol
((cadr q) 3 4)   ➔ Error: can't apply a symbol
(car ''a)        ➔ quote
beta             ➔ (I don (quote t) know)
```

eq?: what is the value returned for the following expressions?

```
(define x 'x)
(define y 'x)
(eq? 'x 'y)   ➔ #f
(eq? x y)     ➔ #t
(eq? 'x y)    ➔ #t
(eq? x 'y)    ➔ #f
(eq? (list 1 2) (list 1 2))     ➔ #f
(equal? (list 1 2) (list 1 2))  ➔ #t
```

`eq?` returns true if the two things **are** the same (don't use with strings or numbers)

`equal?`

- recursively tests if the values are the same
- rule of thumb: returns true if the two things print out the same

Puzzle

Can you write a non-trivial scheme expression that prints itself?

(Hint: Consider how a cell reproduces. It contains its own blueprint (the DNA strand in the nucleus); reproduction involves (a) copying the blueprint, (b) implementing the blueprint. That is, it uses the blueprint twice.)

Hint 2: Create and apply a lambda expression that returns a list structure where the argument is duplicated. The argument should roughly be half of the expression.

memq

`(memq <item> <list>)` will return return a sublist beginning with the first occurrence of the symbol <item>. If the symbol <item> is not contained in the <list>, then memq will return false.

```
(define (memq item lst)
  (cond ((null? lst) false)
        ((eq? item (car lst)) lst)
        (else (memq item (cdr lst)))))
```

tree-equal?

Using eq? write the function tree-equal? that takes two trees of symbols and returns true if the same symbols are arranged in the same structure.

```
(tree-equal? '(this is a list) '(this is a list))
  ➔ #t
(tree-equal? '(this (is a) list) '(this (is a)
  list)) ➔ #t
(tree-equal? '(this is a list) '(this (is a) list))
  ➔ #f
(define (tree-equal? tree1 tree2)
  (cond ((and (null? tree1) (null? tree2)) #t)
        ((and (symbol? tree1) (symbol? tree2))
            (eq? tree1 tree2))
        ((and (pair? tree1) (pair? tree2))
            (and (tree-equal? (car tree1)
                              (car tree2))
                 (tree-equal? (cdr tree1)
                              (cdr tree2))))
         (else #f)))
```

Puzzle

Can you write a non-trivial scheme expression that prints itself?

(Hint: Consider how a cell reproduces. It contains its own blueprint (the DNA strand in the nucleus); reproduction involves (a) copying the blueprint, (b) implementing the blueprint. That is, it uses the blueprint twice.)

Hint 2: Create and apply a lambda expression that returns a list structure where the argument is duplicated. The argument should roughly be half of the expression.

```
((lambda (x)
   (list x (list 'quote x)))
 '(lambda (x)
    (list x (list 'quote x))))
```