```
((lambda (x)
    (* x x) ) 2)
```

prints out 4 because it is using 2 as an input
This is how the let function works, for local definitions

The statement above is equivalent to

```
(let ((x 2))
    (LAMBDA (X)
        (* X X) ) )
```

RECURSION (function composition)

repeat(f, n)   f = function, n = # of times to compose it.

```
(define square
    (lambda (x)
        (* x x) ) )
(define  compose
    (lambda (f g)
        (lambda (x) (f (g x) ) ) ) )
```

the above function returns the f(g)

```
(define repeated
    (lambda (f n)
        (if (> n 0)
            (compose f
                (repeated f ( - n 1) ) )
            (lambda (x) x) ) ) )
```

(repeated square 3) 4)   Composes square 3 times, with input 4

An Idiom for Object Oriented Programming

In OOP, you usually have a constructor for your object, and that object usually has methods.
In scheme, we can fake this.

A constructor will be a function which takes some argument which takes messages and other inputs and produces an output.

In scheme, give constructors names beginning with make_

Suppose in java, we wanted a class which stores an int and allows you to get/set it. In scheme, we could have a function

```
(define my_int
    (make_hold_int 7) )
```

The above function creates an object of type hold_int holding a 7 and gives this object the name my_int

To get the number (my_int get)
7

(my_int set 6)

```
6                               This value will be internally changed

((eqv? Msg ‘distance-left)
        (distance-left player-x player-y edge) )        returns number of visible squares to the left

(define blank-distance-right
        (lambda (x y edge)
                (- edge x) ) )

(define make-blank-game
        (lambda (m)
                (make-flex-game m 1
                        blank-distance-up
                        blank-distance-down
                        blank-distance-left
                        blank-distance-right) ) )
```

TESTING make-blank-game

```
->      (define  maze (make-blank-game 5) )             maze is the variable name, game is 5 x 5 board

->      (maze ‘right!)
#t

->      (maze ‘left!)
#t

->      (maze ‘left!)
#f
```

Page Break