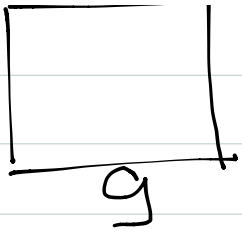
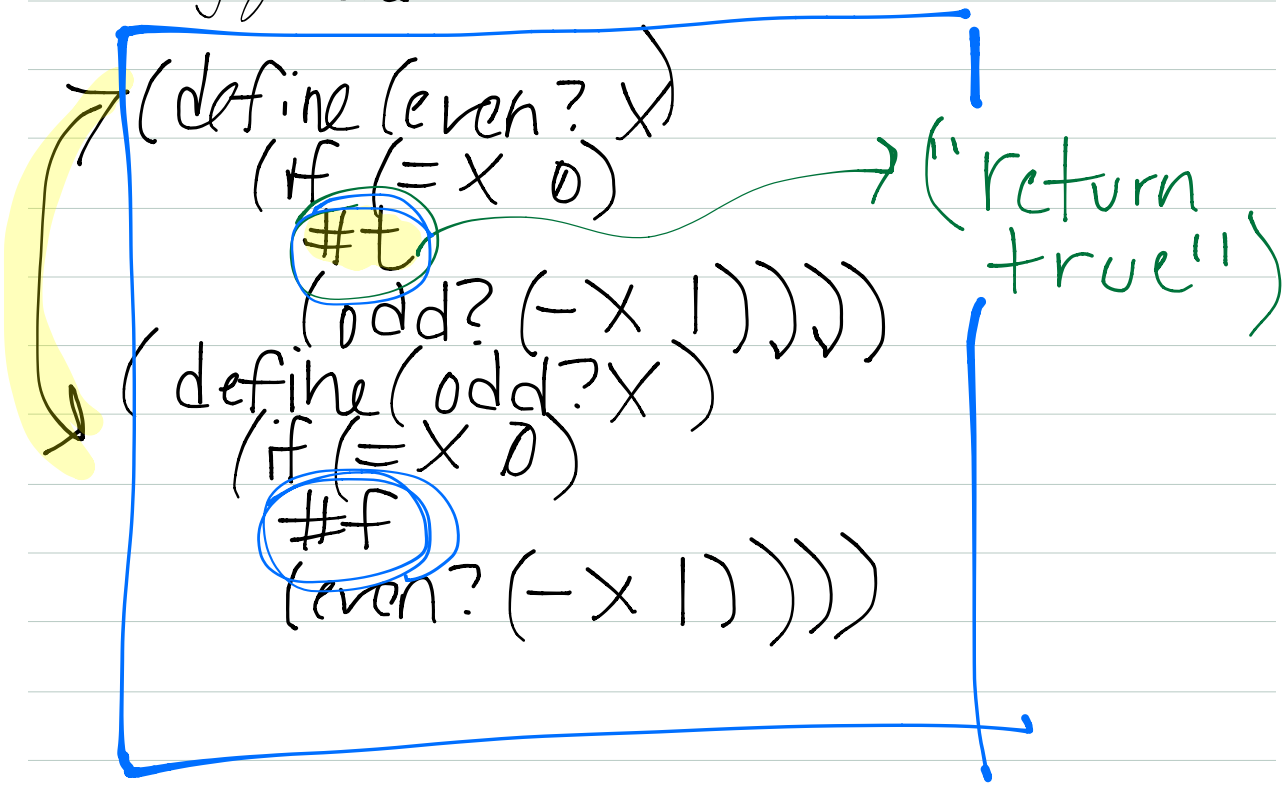
*Review of (sqrt-iter)
↳ Substitution model



* Naming & The Environment: Names of formal parameters don't matter, but the names of external parameters do matter.

---

* Block-structure: when (sqrt 2) is evaluated, 2 is substituted for x in the body of the code.



*Special <if> form:

(if <predicate> <consequent> <alternative>)

*Evaluation Rule:

1. Evaluates <predicate>
2. If eval'n returns #t, entire expression evaluates to what <consequent> evaluates to.
3. Otherwise, evaluates to <alternative> result.

→ In (sqrt-iter 1), 1 is the "base case"

```
(define (factorial n)
  (if (= n 0)
  1
(* n (factorial (- n 1)))))
```

* Order in which you execute commands, matters. The order in which you evaluate expressions, to an extent, doesn't matter- you always get the same answer. Wipe/clean your glasses before you blow your nose.

Time & Space Resources

Time = vertical axis. Space = horizontal axis.

↳ B/c this computational process is linear in time and space. Horizontal/vertical axes grow linearly with the data

- meter[s]

(define (fact-iter prod n)
  [etc...]
;Value: factorial

Iterative version Fibonacci numbers (with ";Value: fib" at the end)

(define (expt b n)
  (cond ((=n 0) 1) → if (n=0), return 1
    ((even? n) (square b
    [...etc]
    (else (.....))) → else, return <x>

(define (matrix-expt b n)
  (cond ((= n 0) 1)
    ((even? n)
      (matrix-square (matrix-expt...
    (else (matrix-*b [...])

Combining clever multiplication w/ 2x2 matrix multiplication....
logarithmic time (see slides)