CS3L:
Introduction to Symbolic Programming

Lecture 2:
Introduction, and Conditionals

Spring 2008

Nate Titterton
nate@berkeley.edu

Announcements

- **Nate's office hours:**
  - Wednesday, 2 – 4
    (except once a month as posted)…
  - 329 Soda

- **Any questions about the course?**
  - Card keys?
  - Working from home?

Schedule

1	Jan 21-25	Lecture: <*holiday*> Lab: (1) Introduction, emacs, unix (2) Words and sentences
2	Jan 28-Feb1	Lecture: Introduction, Review, Conditionals Reading: Simply Scheme, ch. 3-6 Lab: (1) Conditionals and booleans (2) Words/sentences and conditionals *Note: this is a "full" week.*
3	Feb 4-Feb 8	Lecture: Conditionals, Case Studies Reading: "Difference between Dates" case study, in the reader Lab: Explore "Difference between Dates"
4	Feb 11-15	Lecture: Data abstraction in DbD, recursion Reading (Th/Fri): SS ch. 11 Lab: (1) Miniproject 1 (2) Recursion

The *main* topics in the course

(1) Scheme

(2) Recursion

(3) High-order procedures

Also:

… writing “good” programs

How are the labs?

Are you keeping up?
The software ok?

Fall 2005 Final Survey

What advice would you give to students who will take CS3 in future semesters?

Common and helpful responses:

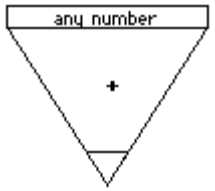
- Go to lab and ask lots of questions; GSI are helpful
- Don't fall behind in lab, catch up as soon as you fall behind
- Do the reading before lab
- prepare to spend a lot of time with the class ***and*** it's not as bad as you might make it out to be.
- find a good partner to do projects and study for exams
- Ask for help
- Focus hard on the first two weeks…

Lab: a look back at day 1

1. Evaluation: from the inside out

```
(+ (* 2 (/ 4 2)) (* (+ 12 1) 2))
```

- How to define functions
- The scheme machine (pictures)



- `sales-tax`, `discount-price`, `selling-price`
- Which single character has changed (to get an unbound error?

```
(define (square x)
        (* x x))
```

6. mystery procedure

```
(define (mystery x)
     (square (+ 1 (truncate (sqrt (- x 1))))) )
```

7. Write the french revolutionary date program

Terminology (from day 1)

- argument
- body
- expression
- evaluation
- input
- placeholder
- procedure
- result

```
> (define (prepend-joe name)
    (word 'joe name))
prepend-joe

> (prepend-joe 'bob)
joebob

> (prepend-joe (word 'j 'o 'e))
joejoe
```

Lab: a look back at day 2

1. **Procedures that take words & sentences**
   `first, last, butfirst, butlast`
2. **Quoting!**
   - names versus things that are named
3. **Constructing words & sentences**
   with `word` and `sentence (se)`

- **Add parens and quotes to get (`def ghi`)**
  `butfirst sentence abc word def ghi`
- **experiment with `appearances`**
- **Evaluation rules with quotes**
- **Packaging information with sentences**
  `(inch-count '(2 3))` ➔ `27`
  `(FR-date 31)` ➔ `(2 1)`

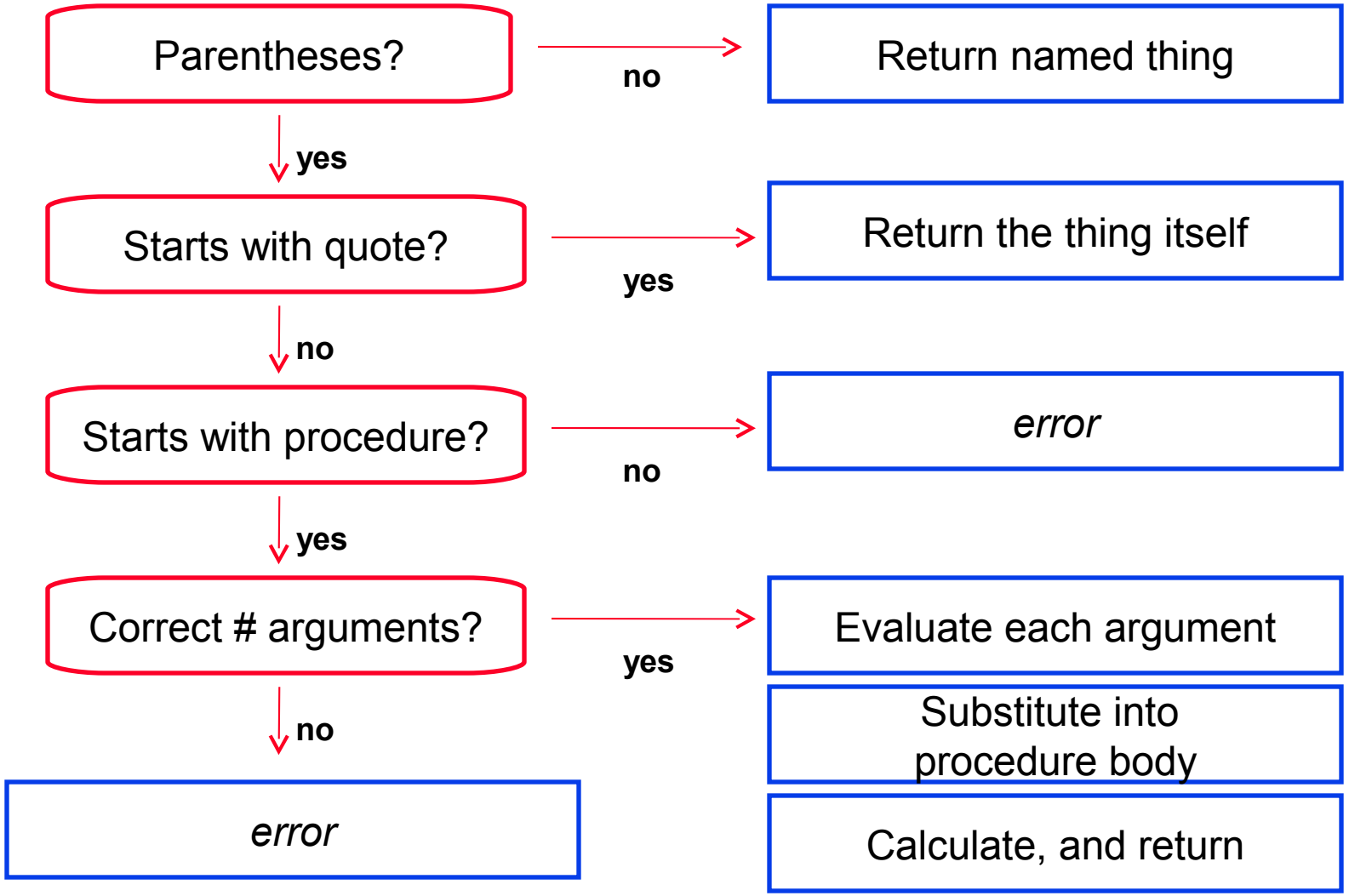
8. **Some common misconceptions**

Quoting

- **Quoting something means treating it *literally*:**
  - **you are interested in the specific thing follows, rather than what is named**
  - **Quoting is a shortcut to putting literal things right in your code. As your programs get bigger, you will do this less and less.**

Quoting is something unique to Scheme

Evaluation of a scheme expression



Common misconceptions

1) All arguments to a procedure get quotes.
2) Sentences don't need to be quoted.
3) Sentences of numbers don't need to be quoted.
4) Words don't need to be quoted.
5) Quotes can go either on the inside or the outside of a sentence.

Some programming

- **“first-two-letters”**
  - takes a word, returns the first two letters (as a two-letter word)

- **“two-first-letters”**
  - takes a sentence of two words, returns the first letter of each (as a two-letter word)

A big idea

- **Data abstraction**
  - Constructors: procedures to make a piece of data
    - `word` and `sentence`
  - Selectors: procedures to return parts of that data piece
    - `first`, `butfirst`, etc.

Coming up: conditionals

- **Conditionals allow programs to do different things depending on data values**
  - ***To make decisions***

- **"Intelligence" depends on this**
  - **it is hard to imagine any interesting program that doesn't do different things depending on what it is given**

Structure of conditionals

```
(if  <true?>                    ;; test
   <do something>               ;; action (if true)
   <do something else>)         ;; action (if false)

(define (smarty x)
   (if (odd? x)
     (se x '(is odd))
     (se x '(is even)))
     )
```

true? or false?

- **We need Booleans: something that represents truth or 'not truth' to the computer:**

```
#t, #f
(odd? 3) ➔ #t
```

  - **in practice, everything is true except `#f`**

```
(if 'joe '(hi joe) '(who are you))
   ➔ (hi joe)
```

  - **`false` (the word with 5 letters) is true!**
    (really, `false` is `#t`)

Predicates

- **Predicates are procedures that return `#t` or `#f`**
  - by convention, their names end with a "?"

```
odd?      (odd? 3) ➔ #t
even?     (even? 3) ➔ #f
vowel?    (vowel? 'a) ➔ #t
          (vowel? (first 'fred)) ➔ #f
sentence?     (sentence? 'fred) ➔ #f
```

`cond` is another conditional form

```
(cond
   (test-1   return-if-test1-true)
   (test-2   return-if-test2-true)
   ...
   (else     return-if-no-other-test-is-true)
    ))
```

`and`, `or` and `not` to modify booleans

- `and` takes any number of arguments, and returns true only if <u>all</u> are true
- `or` takes any number of arguments, and returns true if <u>any</u> are true
- `not` takes a single argument, and returns true only if the argument is false.

```
(if (not (and #t #t #t #f))
    'yes
    'awwwww)      ➔  yes
```

testing

There is much more to programming than writing code

- *Testing* is crucial, and an emphasis of this course

- Analysis
- Debugging
- Maintenance.
- "Design"

- How do you test a conditional?