CS3L: Introduction to Symbolic Programming

Lecture 2:
Introduction, and Conditionals

Spring 2007

Nate Titterton
nate@berkeley.edu

Announcements

- Nate's office hours:
  - Wednesday, 2 - 4
  - 329 Soda
- I'm not hearing about any book or reader supply problems. Yes?
- Any questions about the course?
  - Card keys?
  - Working from home? (check the resources page)

Schedule

1	Jan 15-19	Lecture: <holiday> Lab: (1) Introduction, emacs, unix (2) Words and sentences
2	Jan 22-26	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	Jan 29-Feb 3	Lecture: Conditionals, Case Studies Reading: "Difference between Dates" case study, in the reader Lab: Explore "Difference between Dates"
4	Feb 5-9	Lecture: Data abstraction in DbD Lab: Miniproject 1

How are the labs?

Are you keeping up?

Is the SchemeHandler completely useless, or just mostly useless?

Lab: a look back at day 1

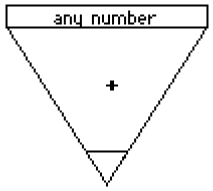
1. **Evaluation: from the inside out**

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

2. **How to define functions**

   (1) `define`, (2) procedure name, (3) parameters, (4) body

3. **The scheme machine (pictures)**



4. `sales-tax`, `discount-price`, `selling-price`

5. **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 french revolutionary date**

Terminology (from lab-session 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)`
4. Add parens and quotes to get `(def ghi)`
   `butfirst sentence abc word def ghi`
5. experiment with `appearances`
6. Evaluation rules with quotes
7. Packaging information with sentences
   `(inch-count '(2 3))` $\rightarrow$ `27`
   `(FR-date 31)` $\rightarrow$ `(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
(and similar languages)

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 <u>Booleans</u>: 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)` $\rightarrow$ `#t`
even?	`(even? 3)` $\rightarrow$ `#f`
vowel?	`(vowel? 'a)` $\rightarrow$ `#t`
	`(vowel? (first 'fred))` $\rightarrow$ `#f`
sentence?	`(sentence? 'fred)` $\rightarrow$ `#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 all are true
- `or` takes any number of arguments, and returns true if any 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?