

## Class 19: Naming Local Values

**Held:** Friday, 22 February 2008

**Summary:** We consider why and how you name values within a procedure, and limit access to those names to within the procedure.

### Related Pages:

- EBoard.
- Lab: Local Bindings.
- Reading: Local Bindings.
- Due: Quiz 4.

### Notes:

- Instructions for running DrFu remotely on a Mac are available on the DrFu web site.
- We'll spend a few minutes making sure that everyone has partners for the homework.
- Does anyone have questions on the homework?
- Reading for Monday: Transforming RGB Colors. Amazingly, that reading is already ready.
- I will be here for my morning office hours, but not my afternoon hours.

### Overview:

- Why name things.
- Naming things with `let`.
- Naming things with `let*`.
- Naming procedures.
- Lab.

## The Problem: Naming Values

- As we've seen in many problems, it helps to name the values that we use within our procedure. Why?
  - It can make the code more readable because the name tells us something about the role the value plays.
  - It can make the code more efficient, because it allows us to avoid recomputing a value.
- Another reason to name things is that we might want to create helper procedures and only make them available to the current procedure.

## Naming Things with `let`

- You name things with `let`.
- `let` has the form

```
(let ((name1 exp1)
      (name2 exp2)
      ...
      (namen expn))
  body)
```

- `let` has the meaning:
  - Evaluate all the expressions.
  - Update the binding table to associate each name with the corresponding value.
  - Evaluate *body* using the updated binding table.
  - Eliminate all the bindings just created.
- You can use `let` in a simple expression:

```
(define values (list 1 4 2 4 1 5 9))
(let ((largest (max values))
      (smallest (min values)))
  (/ (+ largest smallest) 2))
```

- More frequently, we use `let` within a procedure.

## Sequencing Bindings with `let*`

- If we want to bind some things in sequence, we need to use `let*` rather than `let`.
- `let*` has the form

```
(let* ((name1 exp1)
       (name2 exp2)
       ...
       (namen expn))
  body)
```

- `let*` has the meaning:
  - Evaluate *exp<sub>1</sub>*.
  - Update the binding table to associate *name<sub>1</sub>* with that value.
  - Evaluate *exp<sub>2</sub>*.
  - Update the binding table to associate *name<sub>2</sub>* with that value.
  - ...
  - Evaluate *exp<sub>n</sub>*.
  - Update the binding table to associate *name<sub>n</sub>* with that value.
  - Evaluate *body* using the updated binding table.
  - Eliminate all the bindings just created.

## Naming Helper Procedures

- You can also use this technique to name helper procedures.
- Here's an example of the use of a procedure that uses a non-recursive helper procedure that checks whether a value of any type is exact

```
;;; Procedure:
;;;   exact-average
;;; Parameters:
;;;   num1, an exact number
;;;   num2, an exact number
;;; Purpose:
;;;   Average the two numbers.
;;; Produces:
;;;   average, an exact number
;;; Preconditions:
;;;   num1 is an exact number [Verified]
;;;   num2 is an exact number [Verified]
;;; Postconditions:
;;;   Guess.
(define exact-average
  (lambda (num1 num2)
    (let ((verify?
          (lambda (val) (and (number? val) (exact? val))))))
      (cond
        ((not (verify? num1))
         (throw "exact-average" "first parameter is a non-number"))
        ((not (verify? num2))
         (throw "exact-average" "second parameter is a non-number"))
        (else (/ (+ num1 num2) 2))))))
```

---

Copyright © 2007-8 Janet Davis, Matthew Kluber, and Samuel A. Rebelsky. (Selected materials copyright by John David Stone and Henry Walker and used by permission.) This material is based upon work partially supported by the National Science Foundation under Grant No. CCLI-0633090. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.