Lab: Randomness and Simulation

Summary: In this laboratory, we explore Scheme’s random procedure and its use in some simple simulations.

Contents:

- Exercises
  - Exercise 1: Testing random
  - Exercise 2: Rolling Dice
  - Exercise 3: Counting Dice
  - Exercise 4: Testing roll-dice
  - Exercise 5: Flipping Coins
  - Exercise 6: Searching for Paradise
  - Exercise 7: Sevens or Elevens
  - Exercise 8: ... or Doubles
  - Exercise 9: Choosing Names

- Notes
  - Notes on Exercise 2
  - Notes on Problem 7

Exercises

Exercise 1: Testing random

a. Evaluate the expression (random 10) twenty times. What values do you get?

b. Try calling random a few times using 1 as a parameter. What values do you get?

c. Try calling random with -1 as a parameter. What value do you get?

d. Try calling random with other parameters. What effect does the parameter seem to have?

e. What is the largest integer you can provide as a parameter to random?

Exercise 2: Rolling Dice

a. Copy the roll-a-die and roll-dice procedures from the reading. You can also find them at the end of this lab.

b. Using roll-dice, roll ten dice.
c. Using `roll-dice`, roll ten dice. (Yes, this instruction is the same as the previous instruction. You should do it twice.)

d. Did you get the same list of values each time?

e. What other procedures have you encountered that may return different values each time you call them with the same parameters?

**Exercise 3: Counting Dice**

Write a procedure, `(count-odd-rolls n)` that counts the number of odd numbers that come up when rolling *n* six-sided dice.

```lisp
> (count-odd-rolls 10)
7
> (count-odd-rolls 10)
2
> (count-odd-rolls 10)
6
> (count-odd-rolls 10)
5
```

**Exercise 4: Testing `roll-dice`**

Discuss with your partner how you might write a test suite for `roll-dice`. Be prepared to share your answer with the class.

**Exercise 5: Flipping Coins**

a. Write a zero-parameter procedure, `(heads?)` that simulates the flipping of a coin. Heads should return `#t` (which represents "the coin came up heads") half the time and `#f` (which represents "the coin came up tail") about half the time.

```lisp
> (heads?)
#f
> (heads?)
#f
> (heads?)
#t
> (heads?)
#t
> (heads?)
#f
```

b. Write a procedure, `(count-heads n)` that simulates the flipping of *n* coins (using `heads?` to simulate each coin) and returns the number of times the coin is tails.

c. Use `count-heads` to test `heads?` by counting the number of heads you get in 1000 flips.
Exercise 6: Searching for Paradise

a. Write a procedure, (pair-a-dice), that simulates the rolling of two six-sided dice and prints out a pair of the results.

b. Write a procedure, (sum-a-dice), that simulates the rolling of two six-sided dice and then computes their sum.

c. Write a procedure, (count-sevens n) that simulates the rolling of n pairs of dice and counts the number of times the value 7 appears.

Exercise 7: Sevens or Elevens

Consider the problem of rolling a pair of dice n times and counting the number of times that either a 7 or an 11 comes up.

a. What is wrong with the following procedure to accomplish this task?

(define seven-or-11
 (lambda (n)
   (cond ((<= n 0) 0)
         ((or (= (sum-of-dice) 7) (= (sum-of-dice) 11))
          (+ 1 (seven-or-11 (- n 1))))
         (else (seven-or-11 (- n 1))))))

Hint: Try adding a display to sum-of-dice so that you can see how many times sum-of-dice is called. (If you still cannot figure it out after trying that, read the notes on this problem.

b. Write a correct procedure to solve this problem.

Exercise 8: ... or Doubles

Extend your procedure from the previous exercise to count the number of times 7, 11, or “doubles” (two dice with the same value) come up in n rolls.

Exercise 9: Choosing Names

a. Write a procedure, (random-student), that randomly selects the name of a student from this class.

b. Write a procedure, (random-pair), that randomly makes a list of two students from the students of this class.

c. What is the potential problem with me using (random-pair) to select partners from this class?
Notes

Notes on Exercise 2

Just in case you don’t have the reading handy, here’s the code again.

;;; Procedure:
;;;   roll-a-die
;;; Parameters:
;;;   None
;;; Purpose:
;;;   To simulate the rolling of one six-sided die.
;;; Produces:
;;;   An integer between 1 and 6, inclusive.
;;; Preconditions:
;;;   None.
;;; Postconditions:
;;;   Returns an integer between 1 and 6, inclusive.
;;;   It should be difficult (or impossible) to predict which
;;;   number is produced.
(define roll-a-die
  (lambda ()
    (let ((tmp (random 6))) ; tmp is in the range [0 .. 5]
      (+ 1 tmp))))          ; result is in the range [1 .. 6]

;;; Procedure:
;;;   roll-dice
;;; Parameters:
;;;   n, an integer (the number of dice to roll)
;;; Purpose:
;;;   Roll n dice.
;;; Produces:
;;;   A list of integers, each between 1 and 6 (inclusive).
;;; Preconditions:
;;;   n >= 1.
;;; Postconditions:
;;;   Returns a list of length n.
;;;   Each element of the list is between 1 and 6 (inclusive).
;;;   The elements of the list are difficult (or impossible) to predict.
(define roll-dice
  (lambda (n)
    (if (<= n 0)
        null
      ; Otherwise, roll once and then roll n-1 more times.
      (cons (roll-a-die) (roll-dice (- n 1))))))

Notes on Problem 7

a. You may note that the following line involves two calls to the sum-of-dice procedure.
((or (= (sum-of-dice) 7) (= (sum-of-dice) 11)))

Since each call to \texttt{sum-of-dice} involves a roll of the dice, this code says, in effect, “roll the dice and check if the value is seven; if not roll them again and check if the value is eleven”. However, we really want only \textit{one} roll.

b. The solution is to use a \texttt{let} clause.

\begin{verbatim}
(let ((roll (sum-of-dice)))
  ...  
  (or (= roll 7) (= roll 11))
\end{verbatim}