Lab: Simulation

- **Exercises**
  - Exercise 0: Getting Started
  - Exercise 1: Testing `random`
  - Exercise 2: Rolling Dice
  - Exercise 3: Counting Dice
  - Exercise 4: Flipping Coins
  - Exercise 5: Searching for Paradise
  - Exercise 6: Sevens or Elevens
  - Exercise 7: ... or Doubles
  - Exercise 8: Repeated Heads

- **Notes**
  - Notes on Exercise 2

**Exercises**

**Exercise 0: Getting Started**

a. Scan through the reading on simulation.

b. Start DrScheme

**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?

**Exercise 2: Rolling Dice**

a. Copy the `roll-a-die` and `roll-dice` procedures.

b. Using `roll-dice`, roll ten dice.

c. Using `roll-dice`, roll ten dice.
d. Did you get the same list of values each time?

e. What other procedures return different values each time you call them?

**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.

**Exercise 4: Flipping Coins**

a. Write a 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.

b. Write a procedure, `(count-tails 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-tails)` to test `(heads?)` by counting the number of heads you get in 1000 flips.

**Exercise 5: 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 6: 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?

```scheme
(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.
b. Write a correct procedure to solve this problem.

Exercise 7: ... 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 8: Repeated Heads

a. Write a procedure, \((\text{double-heads} \; n)\), which tosses a coin \( n \) times and determines whether a head ever comes up twice in a row.

Do not make a list of flips and then scan through the list. However, you might try adding another parameter which indicates if the previous toss were a head.

b. Write a procedure, \((\text{count-double-heads} \; n)\), which records the number of times a double head is obtained when a coin is tossed \( n \) times. In your counting, you should consider three heads in a row as two double heads.

Notes

Notes on Exercise 2

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

```scheme
;;; 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
  (lambda (n)
    ; If there are no dice left to roll, ...
    (if (<= n 0)
        ; then give an empty list of rolls
        null
      ; Otherwise, roll once and then roll n-1 more times.
      (cons (roll-a-die) (roll-dice (- n 1))))))