Tutorial 2: Creating Functions using the Design Recipe

CS 135 Fall 2007

September 19-21, 2007

This week's tutorial covers how to design, develop, and test Scheme functions. The method we will use is the "Design Recipe", which can be found in module 2 of the lecture notes.

1 Calorie Counter

I'm trying to lose weight, but I'm too lazy to use a calculator to figure out how many calories I've burned. My favorite fitness website tells me that I burn 70 calories for every 10 minutes of walking, and 174 calories for every 10 minutes of running.

Use the Design Recipe to create a Scheme function `calorie-count` which consumes the number of minutes I've spent walking, the number of minutes I've spent running, and the number of calories in the food I've eaten today, and produces a single positive or negative value for the number of calories I've burned today (output-intake).

2 Room Area

You are checking out a possible apartment for next year, but unfortunately the dimensions of the rectangular room are given in feet and inches, and the area of your rug is in square centimeters.

Given that one inch is exactly 2.54 centimeters, write a Scheme function `room-dims-cm` that comsumes four numbers for the lengths of two sides of

the room in feet and inches and produces the area of the room in centimeters squared. Use the Design Recipe to create your procedure.

3 Tasty Beverages at a Party

You occasionally host parties for your friends at school. You always provide tasty beverages for your guests, and you always count how many guests you have. But you'd like to know how many of your friends from CS 135 came. You notice that each of your CS 135 classmates consumes two beverages at the party, while everyone else just has one.

So, given the number of tasty beverages you start out with, the number you have left when the party is over, and the total number of people who attended, write a Scheme function `schemers-at-party` to compute the number of CS 135 students who came to your party. Use the Design Recipe and think carefully about the problem before you start to write any code.

4 Unix Time

On many modern computers, the internal clock of the computer is stored as a single number representing the number of seconds elapsed since the "Unix epoch": Midnight, January 1, 1970. For instance, the first tutorial for this week will be held on September 19 at 11:30am; the Unix time for this is 1190201400.For the following, it will be helpful to know that January 1, 1970 was a Thursday, the 5th day of the week.

4.1 Unix time to Day of Week

Use the Design Recipe to create a function which consumes a time in Unix time and produces an integer from 1 to 7 for the day of the week that time fell on (1 for Sunday, 2 for Monday, 3 for Tuesday, etc.).

Call your function `unixtime->dayofweek`. You may find the built-in Scheme functions `quotient` and `remainder` useful. And you may want to consider using helper functions and/or constants.

4.2 Changing specifications and using a pre-built function

Oh no! After you've already written your program, the requirements have been changed! This often happens in industry and has even been known to happen with CS assignments from time to time.

It turns out no one likes seeing numbers referring to the days of the week. You have been given a function (below) which consumes an integer and returns a string for the corresponding day of the week. You know must use this function so that your `unixtime->dayofweek` returns a string for the day of the week rather than an integer.

You don't have to understand everything in the function definition below. Notice that, because the design recipe has been followed, you can test and use this function without totally understanding how it works.

Also note that you don't need to start from scratch to fix your `unixtime->dayofweek` function. Just follow each step of the design recipe again, updating each part as necessary.

```
;; daynumber->dayname: num -> string
;; Purpose: To convert from a number in the range (1,2,...,7)
;;          to the name of the corresponding weekday.
;; Examples: (daynumber->dayname 6) => "Friday"
(define (daynumber->dayname day)
  (cond [(= day 1) "Sunday"]
        [(= day 2) "Monday"]
        [(= day 3) "Tuesday"]
        [(= day 4) "Wednesday"]
        [(= day 5) "Thursday"]
        [(= day 6) "Friday"]
        [(= day 7) "Saturday"]))
;;Tests:
(daynumber->dayname 6) ;should be "Friday"
(daynumber->dayname 4) ;should be "Wednesday"
(daynumber->dayname 1) ;should be "Sunday"
```