1 Lab 3 Scheme 1

Date: Feb 17, 2022

This document first describes the aims of this lab followed by exercises which need to be performed.

You should perform all exercises using the `racket` dialect of Scheme which is installed on remote.cs. Though this lab assumes that you are performing the exercises within a terminal, you can choose to use `drracket` which provides a GUI interface. If you do use `drracket` you will need to ensure that you capture your output sufficiently to convince the TA that you have indeed completed the lab.

1.1 Aims

The aim of this lab is to introduce you to Scheme. After completing this lab, you should have some familiarity with the following topics:

- Doing arithmetic within Scheme.
- Scheme lists.
- Writing simple functions in Scheme.
- Writing recursive functions in Scheme.

1.2 Exercises

1.2.1 Starting up

Follow the *provided directions* for starting up this lab in a new git `lab3` branch and a new `submit/lab3` directory. Note that this lab does not have an `exercises` directory.

1.2.2 Exercise 1: Arithmetic in Scheme

Fire up racket in the terminal in which you are running the `script` program:

```
$ racket
```

Let's use Scheme as a calculator. Let's calculate the sum of the first few terms of the harmonic series

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5}$$

using the Scheme expression:

```
(+ 1 (/ 1 2) (/ 1 3) (/ 1 4) (/ 1 5))
```

Notice that Scheme leaves the result as an exact rational number. It should be easy to verify the result.

Add the next term to the harmonic series and verify the result. Note that Scheme has automatically reduced the fraction.

Now change the first `1` in the series to a floating point number; note that the result is now a floating point number.

Now write Scheme expressions to:

1. Evaluate the polynomial $3x^3 - 2x^2 + 4x - 1$ at 4. (note that `(expt b e)` yields b^e.
2. Determine the total amount which results after investing a principal of \$1000.00 for 5 years at 5% interest compounded anually. The usual compound interest formula is:

   $$A = P(1 + \frac{r}{n})^{nt}$$

   where A is the amount resulting from the initial principal P, invested at interest rate r which is compounded n times per time period and t is the total number of time periods. In our case, we have $P = 1000$, $r = 0.05$, $n = 1$ and $t = 5$.

   Verify the answer returned by Scheme by comparing it with the answer returned by one of the many compound interest calculators available on the web.

1.3 Exercise 2

In this exercise, we will play with Scheme lists and some list functions:

Let's define some lists:

```
(define list1 '(1 2 3 4 5))
(define list2 '(a b c d e))
```

Run `car` and `cdr` on each list:

```
(car list1)
(cdr list1)
(car list2)
(cdr list2)
```

You can make repeated use of `car` and `cdr` to extract elements from each list. For example to extract the `3` from `list1`, you would use:

`(car (cdr (cdr list1)))`

which as discussed in class can be expressed as `(caddr list1)`.

Append the two lists together:

`(append list1 list2)`

Now look at both `list1` and `list2` and note that they are unchanged; i.e. `append` is non-destructive.

1. Write an expression to extract the `b` from `list2`.
2. Write an expression which results in a list containing the last two elements of `list2`.
3. Write an expression which results in appending the last two elements of `list2` to the last 3 elements of `list1`.

1.3.1 Exercise 3: Defining Simple Scheme Functions

This exercise familiarizes you with the syntax used for defining functions in Scheme.

Let's define a function to calculate the area of a rectangle:

`(define (rect-area width height) (* width height))`

Run the function on a 4x5 rectangle.

(a) Write a function to calculate the perimeter of a rectangle and use that function to calculate the perimeter of a 4x5 rectangle.

(b) Write a function `line-length` to calculate the length of a line given the 2D coordinates of its end-points. Note that the length of a line between (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

   i. Write a version of the function which requires 4 separate parameters x_1, y_1, x_2 and y_2.

   ```
   (line-length 7 4  10 8) => 5
   ```

   Note that Scheme provides a `sqrt` function.

   ii. Write a second version which takes 2 parameters, each of which is a pair giving the x and y coordinates of a point:

   ```
   (line-length '(7 4) '(10 8)) => 5
   ```

1.3.2 Exercise 4: Recursive Scheme Functions

In the subset of Scheme we are using for this lab, we do not use destructive assigment. Hence the only way to do repetition is to use any built-in repetition operations or use recursion. We concentrate on the latter in this exercise.

Let's write a function to multiply all the numbers in a list:

```
(define (mult args)
  (if (null? args)
      1
      (* (car args) (mult (cdr args)))))

(mult '(2 4 8 16)) => 1024
```

Note that we `cdr-` down the list until we find the `()` at the end of the input list and return `mult` of `()` as 1. On the way back up the recursion, we multiply the result of the recursive call with each element.

(a) Write a function `sum` to sum all the elements of its single list argument. What should be the return value when the list is empty?

```
(sum '(1 2 3 4 5)) => 15
```

(b) Two lists having the same length can be zip'd togther to result in a list of 2-element lists where the first element of the pair is an element of the first list and the second element of the pair is the corresponding element of the second list.

Write a function `zip` to zip the contents of two lists together. You may assume that both lists have the same length.

```
(zip '(1 2 3 4) '(a b c d)) =>
   '((1 a) (2 b) (3 c) (4 d))
```