Scheme

1. Special Forms

   (a) \texttt{let} - (\texttt{let bindings body})

   Binds the given bindings for the duration of the body. The bindings are a list of (\textit{name} \textit{value}) pairs. The body consists of one or more expressions which are evaluated in order and the value of last is returned. Let is an example of syntactic sugar:

   \begin{verbatim}
   (let (((arg1 val1) (arg2 val2)) body)
   is equivalent to
   ((lambda (arg1 arg2) body) val1 val2)
   \end{verbatim}

2. Procedures

   (a) \texttt{(map op lst)} – Apply \textit{op} to each element of \textit{lst} in turn and return a list of the results.
   (b) \texttt{(filter pred lst)} – Apply the predicate \textit{pred} to each element of \textit{lst} and return a list of all elements for which the predicate returned true (anything other than \#f).

Class Schedules Data Structures

You’ve been asked to help the registrar manage class schedules, and have started by creating an abstraction for a class’s units, and another to for a class. So far, you have the following:

\begin{verbatim}
(define (make-units C L H)
  (list C L H))
(define get-units-C car)
(define get-units-L cadr)
(define get-units-H caddr)

(define (make-class number units)
  (list number units))
(define get-class-number car)
(define get-class-units cadr)
(define (get-class-total-units class)
  (let ((units (get-class-units class)))
    (+ (get-units-C units)
      (get-units-L units)
      (get-units-H units))))
(define (same-class? c1 c2)
  (= (get-class-number c1) (get-class-number c2)))
\end{verbatim}
Next, you need to define constructors and selectors to form class schedules.

1. Define a constructor `empty-schedule` that returns an empty schedule.

   Order of growth in time & space?

2. Write a selector that when given a class and a schedule, returns a new schedule including the new class:

   (define (add-class class schedule)

   Order of growth in time, space?

3. Write a selector that takes in a schedule and returns the total number of units in that schedule

   (define (total-scheduled-units sched)

   Order of growth in time, space?

4. Write a procedure that drops a particular class from a schedule.

   (define (drop-class sched classnum)

   Order of growth in time, space?

5. Enforce a credit limit by taking in a schedule, and removing classes until the total number of units is less than max-credits.

   (define (credit-limit sched max-credits)

   Order of growth in time, space?
HOPs

(define (make-student number sched-checker)
  (list number (list) sched-checker))

(define get-student-number car)
(define get-student-schedule cadr)
(define get-student-checker caddr)

(define (update-student-schedule student schedule)
  (if ((get-student-checker student) schedule)
      (list (get-student-number student)
            schedule
            (get-student-checker student))
      (error "invalid schedule")))

6. Finish the call to \texttt{make-student} to require the student takes at least 1 class.

\begin{verbatim}
(make-student 575904467
\end{verbatim}

7. Finish the call to \texttt{make-student} to create a first-term freshman (limited to 54 units).

\begin{verbatim}
(make-student 575904467
\end{verbatim}

8. Write a procedure that takes a schedule and returns a list of the class numbers in the schedule. Use \texttt{map}.

\begin{verbatim}
(define (class-numbers schedule)
\end{verbatim}

9. Rewrite \texttt{drop-class} to use \texttt{filter}.
10. Rewrite `credit-limit` to run in $\Theta(n)$ time.