COMP 210, Sprng 2002
Tutorial for First Exam (2/11/2002)

These are my notes on the questions asked and the answers given in the tutorial.

They are fairly complete.

1. Intermezzos on the exam? No

2. How detailed should the templates be?
   - Templates as in class and lab are fine
   - Example: the list template

```
(define (f a-list …)
    (cond
       [(empty? A-list)  … ]
       [(cons?    A-list)
                    … (first a-list) …
                    (f  (rest a-list) … ) .. ]
     ) )
```

3. Definitions and templates for structures and lists

   Example:

```
;; a mogul score is
;;  (make-score name speed air turns)
;; where name is a symbol and speed, air, and turns are numbers
(define-struct score (name speed air turns))

;; Example
   (make-score  'Barhke  5.5  5.3  5.8)

;; Template
(define  ( …  a-score … )
    … (score-name  a-score) …
    … (score-speed  a-score) …
     … (score-air       a-score) …
     … (score-turns  a-score) …
)
```

   And the list example was the class list template

4. Looked at the Fall 2000 Exam

   See the class web site, under the class notes section

5. Program to add two numbers without Scheme's add function

```
(define (add n  m)
   (cond
      [(zero?  n)       m  ]
```

```
        [(positive? n)
              (add1 (add (sub1 n) m))]
    ))
```

6. Are there templates other than struct and list on the test?
   We had some templates for cases, such as

```
    ;; a color is one of
    ;; – green, or
    ;; – purple, or
    ;; – pink
    ;; We will implement colors with a symbol

    ;; Template
    ( define  ( … a-color … )
       (cond
            [(symbol=? a-color ‘green)    …]
            [(symbol=? a-color ‘purple)   …]
            [(symbol=? a-color ‘pink)     …]
       ))
```

7. Can you write Max on a list?
   Introduced the notion of a non-empty-list-of-numbers to deal with the question of taking Max of an empty list

```
;; a non-empty-list-of-numbers is either
;;  –- (cons  f empty), or
;;  –- (cons  f r)
;; where f is a number and r is a non-empty-list-of-numbers
; ; We will use the built-in list constructor

;; Template
(define (f  a-nelon .. )
   (cond
      [(empty? (rest a-nelon))     … (first a-nelon) …]
      [cons?     (rest a-nelon))
            … (first a-nelon)  … (f  (rest a-nelon) … )  … ]
))

;; max-list: nelon → number
;; Purpose: …
(define (max-list a-nelon)
   (cond
      [(empty?  (rest a-nelon)       (first a-nelon)]
      [(cons?     (rest a-nelon)
        (cond
            [(< (first a-nelon) (max-list (rest a-nelon))) (max-list (rest a-nelon))]
            [else                                          (first a-nelon)]
```

)]
))

8. How many examples do we need for a data definition?

   3 to 4 for this exam will be fine …

9. Can we use the **list** constructor?

   When writing out example data, it is fine to use **list**.

   When working in a program, you almost *always* want to use **cons**, because it is the list constructor. **List** actually translates into a series of **cons** invocations.

10. Question on the syntax for accessing structure elements

    See any template for a structure

11. What if I wanted to write a function that took in a list of planes and returned a list of planes whose speed was >= 500 mph?

    Worked the problem based on the standard list template…

```
; fast-planes: list-of-plane → list-of-plane
(define (fast-planes  a-lop)
   [(empty?  a-lop)    empty]
   [(cons?     a-lop)
    (cond
       [(fast-plane?  (first a-alop))        (cons (first a-lop) (fast-planes (rest a-lop)))]
       [else                                 (fast-planes (rest a-lop))]
    )]))

(define (fast-plane? a-plane)
     (>= (plane-speed a-plane) 500)
```

    Follow-up question dealt with equality –- change the helper function

12. Next question