CS 135 Winter 2018
Tutorial 11: Assignment Review and Clarifications
Reminders

• The final exam is on **Monday, April 16 at 9:00 am**. Remember to check your assigned seat and the exam location beforehand.

• Office hours have been rescheduled during the exam period. They will be posted shortly on Piazza and the course website.

• There will be a help session for the final exam. There will be a poll soon on Piazza to determine what time the help session will be held at.
Clicker Question: History

Who invented the lambda calculus?

A  John von Neumann
B  Alonzo Church
C  Kurt Gödel
D  Grace Hopper
E  John McCarthy
Clarification: Explicit Recursion

Explicit recursion describes the recursive code you wrote up to the point where you were introduced to abstract list functions in Module 10. Any function that includes an expression in the body of the function that is an application of the function itself uses explicit recursion. For example:

\[
\text{(define (f x)} \\
\quad (\ldots (f y) \ldots ))
\]

Abstract list functions such as map, filter, foldr, foldl, and build-list are actually working through the data recursively, but the details are hidden in the implementation of these built-in functions. If a question states that you cannot use explicit recursion, you are allowed to use these functions instead.
Clarification: Explicit Recursion

Example: Write a function `sum-lon` that consumes a list of numbers, and produces the sum of all the numbers in the list. You may use explicit recursion.

```scheme
(define (sum-lon lon)
  (cond [(empty? lon) 0]
        [else (+ (first lon) (sum-lon (rest lon)))]))
```

Example: Write the function `sum-lon` as described above. You may not use explicit recursion.

```scheme
(define (sum-lon lon)
  (foldr + 0 lon))
```
Clarification: Functions are Values

In Intermediate Student with lambda, functions are considered first-class values.

They cannot be simplified any further. This is true for built-in functions, user-defined functions, and lambda expressions.

For example, cons, sum-lon on the previous slide, and

\[(\text{lambda} \ (x \ y) \ (\text{cond} \ [(= \ x \ (\text{first} \ y)) \ y] \ [\text{else} \ (\text{cons} \ x \ y)]))\]

are all simple values.

Since they are first-class values, the substitution rules in Module 3 regarding function applications with simple values apply.
Clarification: Functions are Values

CQ: What is the value of the expression below?

\((\text{foldr} \ (\lambda (x \ y) \ (\text{cond} \ [(\equiv x \ (\text{first} \ y)) \ y] \ [\text{else} \ (\text{cons} \ x \ y)])))\)

'(1) '(8 3 3 5)

A  '(1 5 3 8)
B  '(8 3 5 1)
C  '(5 3 8 1)
D  '(1 8 3 5)
E  '(8 3 3 1)
Clarification: Functions are Values

In the previous clicker question, the expression would evaluate to 
’(8 3 5 1) in one step. This is because all the arguments given to 
foldr are considered simple values.

(foldr (lambda (x y)
    (cond [(= x (first y)) y]
    [else (cons x y)])
    ’(1) ’(8 3 3 5))

⇒ ’(8 3 5 1)
Assignment Review

- Assignment 8, Question 3
- Assignment 8, Question 4
- Assignment 10, Question 1
- Any other requests