Group Problem - Stepping with Lambda
Step through the following program:

```scheme
(((lambda (x y) (lambda (x) (* x y))) 5 6) 10)
```
Abstract List Functions
Recall the abstract list functions \( \text{filter} \), \( \text{map} \), \( \text{foldr} \), and \( \text{build-list} \).

\[
\begin{align*}
\text{filter} &: \ (X \rightarrow \text{Bool}) \ (\text{listof} \ X) \rightarrow \ (\text{listof} \ X) \\
\text{map} &: \ (X \rightarrow Y) \ (\text{listof} \ X) \rightarrow \ (\text{listof} \ Y) \\
\text{foldr} &: \ (X \ Y \rightarrow Y) \ Y \ (\text{listof} \ X) \rightarrow Y \\
\text{build-list} &: \ \text{Nat} \ (\text{Nat} \rightarrow X) \rightarrow \ (\text{listof} \ X)
\end{align*}
\]

Use abstract list functions to do the following tasks without explicit recursion:

Add the list of numbers in \( \text{list} \ 1 \ 2 \ 3 \ 4 \ 5 \) :
\( \Rightarrow \) 15

Create a list of odd numbers from 1 to 12:
\( \Rightarrow \) \( \text{list} \ 1 \ 3 \ 5 \ 7 \ 9 \ 11 \)

Double each element in \( \text{list} \ 1 \ 2 \ 3 \ 4 \ 5 \) :
\( \Rightarrow \) \( \text{list} \ 2 \ 4 \ 6 \ 8 \ 10 \)

Keep all the elements in \( \text{list} \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \) that are divisible by 3:
\( \Rightarrow \) \( \text{list} \ 3 \ 6 \)

Group Problem - Stepping with \( \text{foldr} \)
Abstract list functions like \( \text{filter} \), \( \text{map} \), and \( \text{foldr} \) are built-in functions, we evaluate them in one step as long as the arguments are values. Consider the example below:

Step through the following program:

\[
(\text{foldr} (\lambda (\text{itm} \ rr) (\text{cond} [(\text{odd?} \ \text{itm}) \ (\text{cons} \ \text{itm} \ \text{rr})] [\text{else} \ \text{rr}])]) \ \text{empty} \ (\text{list} \ 1 \ 1 \ 2 \ 3 \ 5 \ 8))
\]

\( \Rightarrow \) \( \text{list} \ 1 \ 1 \ 3 \ 5 \)

More Problems - Abstract List Functions
You may omit the design recipe for the following list of exercises.

Write a function called \( \text{factors} \) that consumes a positive integer, \( n \), and produces a list of its factors from 1 to \( n \) inclusive. Do not use explicit recursion.
\( (\text{factors} \ 30) \Rightarrow \) \( \text{list} \ 1 \ 2 \ 3 \ 5 \ 6 \ 10 \ 15 \ 30 \)

Write a function called \( \text{my-min} \) that consumes a non-empty list of numbers and produces the minimum number in the list. Do not use explicit recursion and do not use the built-in function \( \text{min} \).
\( (\text{my-min} \ (\text{list} \ -2.3 \ 2.5 \ 6.5)) \Rightarrow -2.3 \)

Write a function called \( \text{even} \) that consumes a list of natural numbers and counts the even numbers in it.
\( (\text{even} \ (\text{list} \ 1 \ 7 \ 8 \ 4 \ 5 \ 6)) \Rightarrow 3 \)
More Problems - Abstract List Functions

Write a function `make-posns` which consumes two lists of numbers of equal length and create a list of posns, where the ith posn consists of the ith element in the first list and the ith element in the second list.

Hint: The built-in `map` function supports a variable of arguments. Suppose f is a function that consumes n arguments and y1,...,yn are lists of length m, then `(map f y1 ... yn)` produces a list of length m where the ith element of this list is f applied to the ith element of y1...yn in this order.

```scheme
(check-expect (make-posns (list 1 2 3 4 5) (list 6 7 8 9 10))
          (list (make-posn 1 6) (make-posn 2 7)
                (make-posn 3 8) (make-posn 4 9)
                (make-posn 5 10)))
```

Write a function `multi-odds-to` which consumes a natural number and produces the product of all the positive odd numbers that are less or equal to n.

```scheme
(check-expect (multi-odds-to 5) 15)
```

Group Problem: increasing-lists

Without using explicit recursion, write a function called `increasing-lists` that consumes a positive integer n and produces a list of n lists of natural numbers, where the ith list contains the first i + 1 natural numbers.

```scheme
(increasing-lists 1) ⇒ '((0))
(increasing-lists 4) ⇒
  '((0)
   (0 1)
   (0 1 2)
   (0 1 2 3))
```

Group Problem: map-lofn

Write a function `map-lofn` which consumes a `listof Any` and a list of functions. The functions in the consumed list will have the contract `Num → Any`. `map-lofn` produces a list of lists, where each sublist contains the result of applying each function from the consumed list to each number in the consumed `listof Any`. You may not use explicit recursion in your solution.

```scheme
(check-expect (map-lofn (list 3.5 'four 18 "q" 0) (list sqr add1 zero?))
             (list (list 12.25 19 0)
                   (list 4.5 19 1)
                   (list false false true)))
```
Group Problem: separate
Write a function separate that consumes a value called sep and a list. It inserts sep between identical items in the list.
You may not use explicit recursion.

(separate '(a b c) 'x)⇒ '(a b c)
(separate '(a b b c) 'x)⇒ '(a b x b c)
(separate '(a a b b c c) 'x)⇒ '(a x a b x b c x c)

(Optional) Assignment 8 Background
Assignment 8's question 5 is based on Waterloo's coop ranking system. This information may be new for most of you and we will explain the coop ranking algorithm in the following slides. Each student has a list of employers to give ranks to and each employer has a list of students to give ranks to. The preferences that employers and students can give are positive natural numbers. The smaller the preference number is, the more preferred the student/employer is. Employers can choose not to give rankings to students but students must rank all of their employers. Moreover, each employer can only give one student a 1.

Consider the following scenario. Suppose the input ((listof EmpRanking) and (listof StdRanking)) is given as follows:

(define wateremployers
'(("Karen" 2) ("Jimmy" 3) ("Bill" 1) ("Jesse" 2) ("Anne" 2))
("Byron" 3) ("Jimmy" 5) ("Bill" 4) ("Jesse" 1) ("Anne" 2))
("Paul" 4) ("Jimmy" 5) ("Bill" 1) ("Jesse" 2) ("Anne" 2))
("Charlie" 1) ("Jimmy" 5) ("Bill" 2) ("Jesse" 2) ("Anne" 3))
("Rob" 1) ("Jason" 2) ("Jimmy" 5) ("Bill" 4) ("Jesse" 2) ("Anne" 1)))))

(define wateremployees
'(("Jason" 2) ("Byron" 1) ("Paul" 1) ("Charlie" 1) ("Rob" 2))
("Jimmy" 3) ("Karen" 1) ("Byron" 1) ("Paul" 1) ("Charlie" 1) ("Rob" 1))
("Bill" 4) ("Karen" 1) ("Byron" 1) ("Paul" 3) ("Charlie" 2) ("Rob" 2))
("Jesse" 1) ("Karen" 1) ("Byron" 1) ("Paul" 1) ("Charlie" 3) ("Rob" 1))
("Anne" 1) ("Karen" 1) ("Byron" 4) ("Paul" 1) ("Charlie" 1) ("Rob" 2)))
Your program should use the previous information to construct a compound data structure similar to the table below, which has each row corresponding to an employer and each column corresponding to a student. In each cell on row x column y, the number before the slash represents the employer’s preference on the student at column y and the number after the slash represents the student’s preference on the employer at row x. Notice for each employer row, there is only one ‘1’ given to students.

<table>
<thead>
<tr>
<th></th>
<th>Jason</th>
<th>Jimmy</th>
<th>Bill</th>
<th>Jesse</th>
<th>Anne</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karen</td>
<td>2/1</td>
<td>1/1</td>
<td>3/1</td>
<td>3/1</td>
<td>2/1</td>
</tr>
<tr>
<td>Byron</td>
<td>3/1</td>
<td>5/1</td>
<td>4/1</td>
<td>1/1</td>
<td>2/4</td>
</tr>
<tr>
<td>Paul</td>
<td>4/1</td>
<td>5/1</td>
<td>1/3</td>
<td>2/1</td>
<td>2/1</td>
</tr>
<tr>
<td>Charlie</td>
<td>1/1</td>
<td>5/1</td>
<td>2/2</td>
<td>2/3</td>
<td>3/1</td>
</tr>
<tr>
<td>Rob</td>
<td>2/2</td>
<td>5/1</td>
<td>4/2</td>
<td>2/1</td>
<td>1/2</td>
</tr>
</tbody>
</table>

Next, the 2 preference numbers will sum and a random number between 0 and 1 will be generated for each cell to break ties.

<table>
<thead>
<tr>
<th>Jason</th>
<th>Jimmy</th>
<th>Bill</th>
<th>Jesse</th>
<th>Anne</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karen</td>
<td>3.01</td>
<td>2.001</td>
<td>4.019</td>
<td>3.029</td>
</tr>
<tr>
<td>Byron</td>
<td>4.011</td>
<td>6.018</td>
<td>5.021</td>
<td>2.028</td>
</tr>
<tr>
<td>Paul</td>
<td>5.012</td>
<td>6.017</td>
<td>4.022</td>
<td>3.027</td>
</tr>
<tr>
<td>Charlie</td>
<td>2.013</td>
<td>6.016</td>
<td>4.023</td>
<td>5.026</td>
</tr>
<tr>
<td>Rob</td>
<td>4.014</td>
<td>6.015</td>
<td>6.024</td>
<td>3.025</td>
</tr>
</tbody>
</table>

Cell (Row Karen and column Jimmy) has the smallest value, so Karen is matched with Jimmy. And consequently Karen’s row and Jimmy’s column are removed from the table.

Match so far: (Karen, Jimmy)

<table>
<thead>
<tr>
<th>Jason</th>
<th>Jimmy</th>
<th>Bill</th>
<th>Jesse</th>
<th>Anne</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byron</td>
<td>4.011</td>
<td>5.021</td>
<td>2.028</td>
<td>6.888</td>
</tr>
<tr>
<td>Paul</td>
<td>5.012</td>
<td>4.022</td>
<td>3.027</td>
<td>3.999</td>
</tr>
<tr>
<td>Charlie</td>
<td>2.013</td>
<td>4.023</td>
<td>5.026</td>
<td>4.9999</td>
</tr>
<tr>
<td>Rob</td>
<td>4.014</td>
<td>6.024</td>
<td>3.025</td>
<td>3.000001</td>
</tr>
</tbody>
</table>

Cell (Row Charlie and column Jason) has the smallest value, so Charlie is matched with Jason. And consequently Charlie’s row and Jason’s column are removed from the table.

Match so far: (Karen, Jimmy), (Charlie, Jason)
(Optional) Assignment 8 Background

<table>
<thead>
<tr>
<th></th>
<th>Bill</th>
<th>Jesse</th>
<th>Anne</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byron</td>
<td>5.021</td>
<td>2.028</td>
<td>6.888</td>
</tr>
<tr>
<td>Paul</td>
<td>4.022</td>
<td>3.027</td>
<td>3.999</td>
</tr>
<tr>
<td>Rob</td>
<td>6.024</td>
<td>3.025</td>
<td>3.000001</td>
</tr>
</tbody>
</table>

Cell (Row Byron and column Jesse) has the smallest value, so Byron is matched with Jesse. And consequently Byron’s row and Jesse’s column are removed from the table.

Match so far: (Karen, Jimmy), (Charlie, Jason), (Byron, Jesse)

<table>
<thead>
<tr>
<th></th>
<th>Bill</th>
<th>Anne</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paul</td>
<td>4.022</td>
<td>3.999</td>
</tr>
<tr>
<td>Rob</td>
<td>6.024</td>
<td>3.000001</td>
</tr>
</tbody>
</table>

Cell (Row Rob and column Anne) has the smallest value, so Rob is matched with Anne.

Match so far: (Karen, Jimmy), (Charlie, Jason), (Byron, Jesse), (Rob, Anne)

Lastly, Paul is matched with Bill

Match completed: (Karen, Jimmy), (Charlie, Jason), (Byron, Jesse), (Rob, Anne), (Paul, Bill)

Note in this example each student is matched with a corresponding employer. However, we could also have situations where the students outnumber the employers and vice-versa.
Review: Lambda

\[
((\text{lambda} \ (x_1 \ldots x_n) \ \text{exp}) \ v_1 \ldots v_n) \Rightarrow \text{exp}'
\]

where \(\text{exp}'\) is \(\text{exp}\) with all occurrences of \(x_1\) replaced by \(v_1\), all occurrences of \(x_2\) replaced by \(v_2\), and so on.

For example, the next step here would be:

\[
((\text{lambda} \ (x \ y) \ (\ast \ (\ + \ y \ 4) \ x)) \ 5 \ 6) \Rightarrow \ (\ast \ (\ + \ 6 \ 4) \ 5)
\]

Review: foldr

Recall that foldr is the built-in function that abstracts recursion on lists, where the first element of the list is combined with the recursive call on the rest.

Here is an implementation of foldr as my-foldr:

\[
\text{;; my-foldr: (X Y \to Y) Y (listof X) \to Y}
\]

\[
\text{(define (my-foldr combine base lst)}
\]

\[
\text{\quad (cond}
\]

\[
\text{\quad \quad [empty? lst] base}
\]

\[
\text{\quad \quad [else (combine (first lst)
}\]

\[
\text{\quad \quad \quad (my-foldr combine base (rest lst))))))}
\]

\[
\text{)}
\]

\[
\text{)}
\]

\[
\text{)}
\]

\[
\text{)}
\]