Final Review

The following worksheet is final review! It covers various topics that have been seen throughout the semester.

Your TA will not be able to get to all of the problems on this worksheet so feel free to work through the remaining problems on your own. Bring any questions you have to office hours or post them on piazza.

Good luck on the final and congratulations on making it to the last discussion of CS61A!

Recursion

Q1: Paths List

(Adapted from Fall 2013) Fill in the blanks in the implementation of `paths`, which takes as input two positive integers `x` and `y`. It returns a list of paths, where each path is a list containing steps to reach `y` from `x` by repeated incrementing or doubling. For instance, we can reach 9 from 3 by incrementing to 4, doubling to 8, then incrementing again to 9, so one path is `[3, 4, 8, 9]`.

```python
def paths(x, y):
    """Return a list of ways to reach y from x by repeated incrementing or doubling.
    >>> paths(3, 5)
    [[3, 4, 5]]
    >>> sorted(paths(3, 6))
    [[3, 4, 5, 6], [3, 6]]
    >>> sorted(paths(3, 9))
    [[3, 4, 5, 6, 7, 8, 9], [3, 4, 8, 9], [3, 6, 7, 8, 9]]
    >>> paths(3, 3) # No calls is a valid path
    [[3]]
    >>> paths(5, 3) # There is no valid path from x to y
    []
    ""
    if _______________________________
        return ___________________________
    elif _______________________________
        return ___________________________
    else:
        a = _______________________________
        b = _______________________________
        return ___________________________
```
Mutation

Q2: Reverse

Write a function that reverses the given list. Be sure to mutate the original list. This is practice, so don’t use the built-in reverse function!

```python
def reverse(lst):
    """Reverses lst using mutation."

    >>> original_list = [5, -1, 29, 0]
    >>> reverse(original_list)
    >>> original_list
    [0, 29, -1, 5]
    >>> odd_list = [42, 72, -8]
    >>> reverse(odd_list)
    >>> odd_list
    [-8, 72, 42]
    """
    "*** YOUR CODE HERE ***"

# You can use more space on the back if you want
```

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.
Trees

Q3: Widest Level

Write a function that takes a Tree object and returns the elements at the depth with the most elements.

In this problem, you may find it helpful to use the second optional argument to sum, which provides a starting value. All items in the sequence to be summed will be concatenated to the starting value. By default, start will default to 0, which allows you to sum a sequence of numbers. We provide an example of sum starting with a list, which allows you to concatenate items in a list.

def widest_level(t):
    
    >>> sum([[1], [2]], [])
    [1, 2]
    >>> t = Tree(3, [Tree(1, [Tree(1), Tree(5)]),
                   ...           Tree(4, [Tree(9, [Tree(2)])])])
    >>> widest_level(t)
    [1, 5, 9]
    
    levels = []
    x = [t]
    while _____________:
        _____________ = sum(___________________________, [])
        return max(levels, key=___________________________)

Q4: In-order traversal

Write a function that returns a generator that generates an “in-order” traversal, in which we yield the value of every node in order from left to right, assuming that each node has either 0 or 2 branches.
def in_order_traversal(t):
    """
    Generator function that generates an "in-order" traversal, in which we
    yield the value of every node in order from left to right, assuming that each node
    has either 0 or 2 branches.

    For example, take the following tree t:
    1
    2  3
    4  5
    6  7

    We have the in-order-traversal 4, 2, 6, 5, 7, 1, 3

    >>> t = Tree(1, [Tree(2, [Tree(4), Tree(5, [Tree(6), Tree(7)])])], Tree(3))
    >>> list(in_order_traversal(t))
    [4, 2, 6, 5, 7, 1, 3]
    """

    "*** YOUR CODE HERE ***"

# You can use more space on the back if you want

Linked Lists

Q5: Deep Map

Implement deep_map, which takes a function f and a link. It returns a new linked list with the same structure as link, but with f applied to any element within link or any Link instance contained in link.

The deep_map function should recursively apply fn to each of that Link's elements rather than to that Link itself.

Hint: You may find the built-in isinstance function for checking if something is an instance of an object. For example: >>> isinstance([1, 2, 3], list) True >>> isinstance(Link(1), Link) True >>> isinstance(Link(1, Link(2)), list) False

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.
def deep_map(f, link):
    """Return a Link with the same structure as link but with fn mapped over its elements. If an element is an instance of a linked list, recursively apply fn inside that linked list as well.
    """

    >>> s = Link(1, Link(Link(2, Link(3)), Link(4)))
    >>> print(deep_map(lambda x: x * x, s))
    <1 <4 9> 16>
    >>> print(s)  # unchanged
    <1 <2 3> 4>
    >>> print(deep_map(lambda x: 2 * x, Link(s, Link(Link(Link(5))))))
    <<2 <4 6> 8 <<10>>>
    """
    "*** YOUR CODE HERE ***"

Generators

Q6: Repeated

Write a generator function that yields functions that are repeated applications of a one-argument function f. The first function yielded should apply f 0 times (the identity function), the second function yielded should apply f once, etc.
```python
def repeated(f):
    """
    >>> double = lambda x: 2 * x
    >>> funcs = repeated(double)
    >>> identity = next(funcs)
    >>> double = next(funcs)
    >>> quad = next(funcs)
    >>> oct = next(funcs)
    >>> quad(1)
    4
    >>> oct(1)
    8
    >>> [g(1) for _, g in ...
    ...  zip(range(5), repeated(lambda x: 2 * x))]
    [1, 2, 4, 8, 16]
    """

    g = ________________________________
    while True:
        ______________________________
        ______________________________
```

*Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.*
Scheme

Q7: Group by Non-Decreasing

Define a function `nondecreaselist`, which takes in a scheme list of numbers and outputs a list of lists, which overall has the same numbers in the same order, but grouped into lists that are non-decreasing.

For example, if the input is a stream containing elements

```
(1 2 3 4 1 2 3 4 1 1 1 2 1 1 0 4 3 2 1)
```

the output should contain elements

```
((1 2 3 4) (1 2 3 4) (1 1 1 2) (1 1) (0 4) (3) (2) (1))
```

Note: The skeleton code is just a suggestion; feel free to use your own structure if you prefer.

```
(define (nondecreaselist s)
  (if _____________________________
      _____________________________
      (let ((rest ____________________________ ))
        (if _____________________________
            _____________________________
            (cons _____________________________ rest)
            (cons _____________________________ (cdr rest))
        )
      )
  )

(expect (nondecreaselist '(1 2 3 1 2 3)) ((1 2 3) (1 2 3)))
(expect (nondecreaselist '(1 2 3 4 1 2 3 4 1 1 2 1 1 0 4 3 2 1))
  ((1 2 3 4) (1 2 3 4) (1 1 1 2) (1 1) (0 4) (3) (2) (1)))
```
Programs as Data

Q8: Or with Multiple Args

Recall make-or from Discussion 11. Implement make-long-or, which returns, as a list, a program that takes in any number of expressions and or's them together (applying short-circuiting rules). This procedure should do this without using the or special form. Unlike the make-or procedure from Discussion 11, the arguments will be passed in as a list named args.

The behavior of the or procedure is specified by the following doctests:

```scheme
(scm> (define or-program (make-long-or '((print 'hello) (/ 1 0) 3 #f)))
or-program
(scm> (eval or-program))
hello
(scm> (eval (make-long-or '((= 1 0) #f (+ 1 2) (print 'goodbye)))))
3
(scm> (eval (make-long-or '((> 3 1))))))
#t
(scm> (eval (make-long-or '())))
#f

(define (make-long-or args)
  'YOUR-CODE-HERE)
```

# You can use more space on the back if you want

SQL

The following questions will refer to two tables: - records: a table that stores information about the employees at a small company - meetings: a table which records the divisional meetings at the company

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.
<table>
<thead>
<tr>
<th>Name</th>
<th>Division</th>
<th>Title</th>
<th>Salary</th>
<th>Supervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben Bitdiddle</td>
<td>Computer</td>
<td>Wizard</td>
<td>60000</td>
<td>Oliver Warbucks</td>
</tr>
<tr>
<td>Alyssa P Hacker</td>
<td>Computer</td>
<td>Programmer</td>
<td>40000</td>
<td>Ben Bitdiddle</td>
</tr>
<tr>
<td>Cy D Feat</td>
<td>Computer</td>
<td>Programmer</td>
<td>35000</td>
<td>Ben Bitdiddle</td>
</tr>
<tr>
<td>Lem E Tweakit</td>
<td>Computer</td>
<td>Technician</td>
<td>25000</td>
<td>Ben Bitdiddle</td>
</tr>
<tr>
<td>Louis Reasoner</td>
<td>Computer</td>
<td>Programmer Trainee</td>
<td>30000</td>
<td>Alyssa P Hacker</td>
</tr>
<tr>
<td>Oliver Warbucks</td>
<td>Administration</td>
<td>Big Wheel</td>
<td>150000</td>
<td>Oliver Warbucks</td>
</tr>
<tr>
<td>Eben Scrooge</td>
<td>Accounting</td>
<td>Chief Accountant</td>
<td>75000</td>
<td>Oliver Warbucks</td>
</tr>
<tr>
<td>Robert Cratchet</td>
<td>Accounting</td>
<td>Scrivener</td>
<td>18000</td>
<td>Eben Scrooge</td>
</tr>
</tbody>
</table>

meetings

<table>
<thead>
<tr>
<th>Division</th>
<th>Day</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>Monday</td>
<td>9am</td>
</tr>
<tr>
<td>Computer</td>
<td>Wednesday</td>
<td>4pm</td>
</tr>
<tr>
<td>Administration</td>
<td>Monday</td>
<td>11am</td>
</tr>
<tr>
<td>Administration</td>
<td>Wednesday</td>
<td>4pm</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Q9: Oliver Employee Meetings**

Write a query that outputs the meeting days and times of all employees directly supervised by Oliver Warbucks.

SELECT "YOUR CODE HERE"

# You can use more space on the back if you want

**Q10: Different Division**

Write a query that outputs the names of employees whose supervisor is in a different division.

SELECT "YOUR CODE HERE"