Lab 14 Solutions **lab14.zip (lab14.zip)**

Solution Files

This lab has many files. Remember to write in `lab14.scm` for the Scheme questions, `lab14.sql` for the SQL questions, `lab14.lark` for the BNF question, and `lab14.py` for all other questions.

Required Questions

Trees

Q1: Prune Min

Write a function that prunes a `Tree` `t` mutatively. `t` and its branches always have zero or two branches. For the trees with two branches, reduce the number of branches from two to one by keeping the branch that has the smaller label value. Do nothing with trees with zero branches.

Prune the tree in a direction of your choosing (top down or bottom up). The result should be a linear tree.

```
def prune_min(t):
    """Prune the tree mutatively.

    >>> t1 = Tree(6)
    >>> prune_min(t1)
    >>> t1
    Tree(6)
    >>> t2 = Tree(6, [Tree(3), Tree(4)])
    >>> prune_min(t2)
    >>> t2
    Tree(6, [Tree(3)])
    >>> t3 = Tree(6, [Tree(3, [Tree(1), Tree(2)]), Tree(5, [Tree(3), Tree(4)])])
    >>> prune_min(t3)
    >>> t3
    Tree(6, [Tree(3, [Tree(1)])])
    """
    if t.branches == []:
        return
    prune_min(t.branches[0])
    prune_min(t.branches[1])
    if (t.branches[0].label > t.branches[1].label):
        t.branches.pop(0)
    else:
        t.branches.pop(1)
    return # return statement to block alternate from running

# Alternate solution
    if t.is_leaf():
        return
    remove_ind = int(t.branches[0].label < t.branches[1].label)
    t.branches.pop(remove_ind)
    prune_min(t.branches[0])
```

Use Ok to test your code:

```
python3 ok -q prune_min
```

Scheme

Q2: Split

Implement `split-at`, which takes a list `lst` and a non-negative number `n` as input and returns a pair `new` such that `(car new)` is the first `n` elements of `lst` and `(cdr new)` is the remaining elements of `lst`. If `n` is greater than the length of `lst`, `(car new)` should be `lst` and `(cdr new)` should be `nil`.

```
scm> (car (split-at '(2 4 6 8 10) 3))
(2 4 6)
scm> (cdr (split-at '(2 4 6 8 10) 3))
(8 10)
```

```
(define (split-at lst n)
  (cond ((= n 0) (cons nil lst))
        ((null? lst) (cons lst nil))
        (else (let ((rec (split-at (cdr lst) (- n 1))))
                (cons (cons (car lst) (car rec)) (cdr rec)))))
)
```

Use Ok to test your code:

```
python3 ok -q split-at
```

Q3: Compose All

Implement `compose-all`, which takes a list of one-argument functions and returns a one-argument function that applies each function in that list in turn to its argument. For example, if `func` is the result of calling `compose-all` on a list of functions `(f g h)`, then `(func x)` should be equivalent to the result of calling `(h (g (f x)))`.

```
scm> (define (square x) (* x x))
square
scm> (define (add-one x) (+ x 1))
add-one
scm> (define (double x) (* x 2))
double
scm> (define composed (compose-all (list double square add-one)))
composed
scm> (composed 1)
5
scm> (composed 2)
17
```

```
(define (compose-all funcs)
  (lambda (x)
    (if (null? funcs)
        x
        ((compose-all (cdr funcs)) ((car funcs) x))))
)
```

Use Ok to test your code:

```
python3 ok -q compose-all
```

Regex

Q4: Address First Line

Write a regular expression that parses strings and returns any expressions which contain the first line of a US mailing address.

US mailing addresses typically contain a block number, which is a sequence of 3-5 digits, following by a street name. The street name can consist of multiple words but will always end with a street type abbreviation, which itself is a sequence of 2-5 English letters. The street name can also optionally start with a cardinal direction ("N", "E", "W", "S"). Everything should be properly capitalized.

Proper capitalization means that the first letter of each name is capitalized. It is fine to have things like "WeirdCApitalization" match.

See the doctests for some examples.

```
def address_oneline(text):
    """
    Finds and returns expressions in text that represent the first line
    of a US mailing address.

    >>> address_oneline("110 Sproul Hall, Berkeley, CA 94720")
    ['110 Sproul Hall']
    >>> address_oneline("What's at 39177 Farwell Dr? Is there a 39177 Nearwell Dr?")
    ['39177 Farwell Dr', '39177 Nearwell Dr']
    >>> address_oneline("I just landed at 780 N McDonnell Rd, and I need to get to 1880-is
    ['780 N McDonnell Rd']
    >>> address_oneline("123 Le Roy Ave")
    ['123 Le Roy Ave']
    >>> address_oneline("110 Unabbreviated Boulevard")
    []
    >>> address_oneline("790 lowercase St")
    []
    """
    block_number = r"\d{3,5}"
    cardinal_dir = r"(?:[NEWS] )?" # whitespace is important!
    street = r"(?:[A-Z][A-Za-z]+ )+"
    type_abbr = r"[A-Z][a-z]{1,4}\b"
    street_name = f"{cardinal_dir}{street}{type_abbr}"
    return re.findall(f"{block_number} {street_name}", text)
```

Use Ok to test your code:

```
python3 ok -q address_oneline
```

SQL

In each question below, you will define a new table based on the following tables. The first defines the names, opening, and closing hours of great pizza places in Berkeley. The second defines typical meal times (for college students). A pizza place is open for a meal if the meal time is at or within the `open` and `close` times.

Your tables should still perform correctly even if the values in these tables were to change. Don't just hard-code the output to each query.

Q5: Opening Times

You'd like to have lunch before 1pm. Create a `opening` table with the names of all Pizza places that open before 1pm, listed in reverse alphabetical order.

```
-- Pizza places that open before 1pm in alphabetical order
create table opening as
  select name from pizzas WHERE open < 13 order by name DESC;
```

Use Ok to test your code:

```
python3 ok -q opening
```

Q6: Double Pizza

If two meals are more than 6 hours apart, then there's nothing wrong with going to the same pizza place for both, right? Create a `double` table with three columns. The first columns is the earlier meal, the second is the later meal, and the third is the name of a pizza place. Only include rows that describe two meals that are **more than 6 hours apart** and a pizza place that is open for both of the meals. The rows may appear in any order.

```
-- Two meals at the same place
create table double as
  select a.meal, b.meal, name
         from meals as a, meals as b, pizzas
         where open <= a.time and a.time <= close and
               open <= b.time and b.time <= close and
               b.time > a.time + 6;
```

```
-- Example:
select * from double where name="Sliver";
-- Expected output:
--   breakfast|dinner|Sliver
```

Use Ok to test your code:

```
python3 ok -q double
```

Recommended Questions

The following problems are not required for credit on this lab but may help you prepare for the final.

Objects

Let's implement a game called Election. In this game, two players compete to try and earn the most votes. Both players start with 0 votes and 100 popularity.

The two players alternate turns, and the first player starts. Each turn, the current player chooses an action. There are two types of actions:

- The player can debate, and either gain or lose 50 popularity. If the player has popularity `p1` and the other player has popularity `p2`, then the probability that the player gains 50 popularity is `max(0.1, p1 / (p1 + p2))` Note that the `max` causes the probability to never be lower than 0.1.
- The player can give a speech. If the player has popularity `p1` and the other player has popularity `p2`, then the player gains `p1 // 10` votes and popularity and the other player loses `p2 // 10` popularity.

The game ends when a player reaches 50 votes, or after a total of 10 turns have been played (each player has taken 5 turns). Whoever has more votes at the end of the game is the winner!

Q7: Player

First, let's implement the `Player` class. Fill in the `debate` and `speech` methods, that take in another `Player` `other`, and implement the correct behavior as detailed above. Here are two additional things to keep in mind:

- In the `debate` method, you should call the provided `random` function, which returns a random float between 0 and 1. The player should gain 50 popularity if the random number is smaller than the probability described above, and lose 50 popularity otherwise.
- Neither players' popularity should ever become negative. If this happens, set it equal to 0 instead.

```
### Phase 1: The Player Class
class Player:
    """
    >>> random = make_test_random()
    >>> p1 = Player('Hill')
    >>> p2 = Player('Don')
    >>> p1.popularity
    100
    >>> p1.debate(p2)  # random() should return 0.0
    >>> p1.popularity
    150
    >>> p2.popularity
    100
    >>> p2.votes
    0
    >>> p2.speech(p1)
    >>> p2.votes
    10
    >>> p2.popularity
    110
    >>> p1.popularity
    135


    >>> # Additional correctness tests
    >>> p1.speech(p2)
    >>> p1.votes
    13
    >>> p1.popularity
    148
    >>> p2.votes
    10
    >>> p2.popularity
    99
    >>> for _ in range(4):  # 0.1, 0.2, 0.3, 0.4
    ...     p1.debate(p2)
    >>> p2.debate(p1)
    >>> p2.popularity
    49
    >>> p2.debate(p1)
    >>> p2.popularity
    0
    """
    def __init__(self, name):
        self.name = name
        self.votes = 0
        self.popularity = 100

    def debate(self, other):
        prob = max(0.1, self.popularity / (self.popularity + other.popularity))
        if random() < prob:
            self.popularity += 50
        else:
            self.popularity = max(0, self.popularity - 50)

    def speech(self, other):
        self.votes += self.popularity // 10
        self.popularity += self.popularity // 10
        other.popularity -= other.popularity // 10

    def choose(self, other):
        return self.speech
```

Use Ok to test your code:

```
python3 ok -q Player
```

Q8: Game

Now, implement the `Game` class. Fill in the `play` method, which should alternate between the two players, starting with `p1`, and have each player take one turn at a time. The `choose` method in the `Player` class returns the method, either `debate` or `speech`, that should be called to perform the action.

In addition, fill in the `winner` property method, which should return the player with more votes, or `None` if the players are tied.

```
### Phase 2: The Game Class
class Game:
    """
    >>> p1, p2 = Player('Hill'), Player('Don')
    >>> g = Game(p1, p2)
    >>> winner = g.play()
    >>> p1 is winner
    True



    >>> # Additional correctness tests
    >>> winner is g.winner
    True
    >>> g.turn
    10
    >>> p1.votes = p2.votes
    >>> print(g.winner)
    None
    """
    def __init__(self, player1, player2):
        self.p1 = player1
        self.p2 = player2
        self.turn = 0

    def play(self):
        while not self.game_over:
            if self.turn % 2 == 0:
                curr, other = self.p1, self.p2
            else:
                curr, other = self.p2, self.p1
            curr.choose(other)(other)
            self.turn += 1
        return self.winner

    @property
    def game_over(self):
        return max(self.p1.votes, self.p2.votes) >= 50 or self.turn >= 10

    @property
    def winner(self):
        if self.p1.votes > self.p2.votes:
            return self.p1
        elif self.p2.votes > self.p1.votes:
            return self.p2
        else:
            return None
```

Use Ok to test your code:

```
python3 ok -q Game
```

Q9: New Players

The `choose` method in the `Player` class is boring, because it always returns the `speech` method. Let's implement two new classes that inherit from `Player`, but have more interesting `choose` methods.

Implement the `choose` method in the `AggressivePlayer` class, which returns the `debate` method if the player's popularity is less than or equal to `other`'s popularity, and `speech` otherwise. Also implement the `choose` method in the `CautiousPlayer` class, which returns the `debate` method if the player's popularity is 0, and `speech` otherwise.

```
### Phase 3: New Players
class AggressivePlayer(Player):
    """
    >>> random = make_test_random()
    >>> p1, p2 = AggressivePlayer('Don'), Player('Hill')
    >>> g = Game(p1, p2)
    >>> winner = g.play()
    >>> p1 is winner
    True


    >>> # Additional correctness tests
    >>> p1.popularity = p2.popularity
    >>> p1.choose(p2) == p1.debate
    True
    >>> p1.popularity += 1
    >>> p1.choose(p2) == p1.debate
    False
    >>> p2.choose(p1) == p2.speech
    True
    """
    def choose(self, other):
        if self.popularity <= other.popularity:
            return self.debate
        else:
            return self.speech

class CautiousPlayer(Player):
    """
    >>> random = make_test_random()
    >>> p1, p2 = CautiousPlayer('Hill'), AggressivePlayer('Don')
    >>> p1.popularity = 0
    >>> p1.choose(p2) == p1.debate
    True
    >>> p1.popularity = 1
    >>> p1.choose(p2) == p1.debate
    False


    >>> # Additional correctness tests
    >>> p2.choose(p1) == p2.speech
    True
    """
    def choose(self, other):
        if self.popularity == 0:
            return self.debate
        else:
            return self.speech
```

Use Ok to test your code:

```
python3 ok -q AggressivePlayer
python3 ok -q CautiousPlayer
```

Tree Recursion

Q10: Add trees

Define the function `add_trees`, which takes in two trees and returns a new tree where each corresponding node from the first tree is added with the node from the second tree. If a node at any particular position is present in one tree but not the other, it should be present in the new tree as well.

Hint: You may want to use the built-in zip function to iterate over multiple sequences at once.

```
def add_trees(t1, t2):
    """
    >>> numbers = Tree(1,
    ...                 [Tree(2,
    ...                       [Tree(3),
    ...                        Tree(4)]),
    ...                  Tree(5,
    ...                       [Tree(6,
    ...                             [Tree(7)]),
    ...                        Tree(8)])])
    >>> print(add_trees(numbers, numbers))
    2
      4
        6
        8
      10
        12
          14
        16
    >>> print(add_trees(Tree(2), Tree(3, [Tree(4), Tree(5)])))
    5
      4
      5
    >>> print(add_trees(Tree(2, [Tree(3)]), Tree(2, [Tree(3), Tree(4)])))
    4
      6
      4
    >>> print(add_trees(Tree(2, [Tree(3, [Tree(4), Tree(5)])]), \
    Tree(2, [Tree(3, [Tree(4)]), Tree(5)])))
    4
      6
        8
        5
      5
    """
    if not t1:
        return t2
    if not t2:
        return t1
    new_label = t1.label + t2.label
    t1_branches, t2_branches = list(t1.branches), list(t2.branches)
    length_t1, length_t2 = len(t1_branches), len(t2_branches)
    if length_t1 < length_t2:
        t1_branches += [None for _ in range(length_t1, length_t2)]
    elif length_t1 > length_t2:
        t2_branches += [None for _ in range(length_t2, length_t1)]
    return Tree(new_label, [add_trees(branch1, branch2) for branch1, branch2 in zip(t1_bra
```

Use Ok to test your code:

```
python3 ok -q add_trees
```

Walkthrough

YouTube link (https://youtu.be/pbMeCRUU7yw?t=37m8s)

Linked Lists

Folding Linked Lists

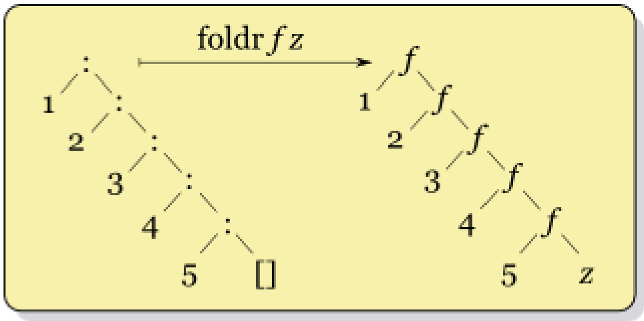
When we write recursive functions acting on `Link` s, we often find that they have the following form:

```
def func(link):
    if link is Link.empty:
        return <Base case>
    else:
        return <Expression involving func(link.rest)>
```

In the spirit of abstraction, we want to factor out this commonly seen pattern. It turns out that we can define an abstraction called `fold` that do this.

A linked list can be represented as a series of `Link` constructors, where `Link.rest` is either another linked list or the empty list.

We represent such a list in the diagram below:



In this diagram, the recursive list

```
Link(1, Link(2, Link(3, Link(4,Link(5)))))
```

is represented with `:` as the constructor and `[]` as the empty list.

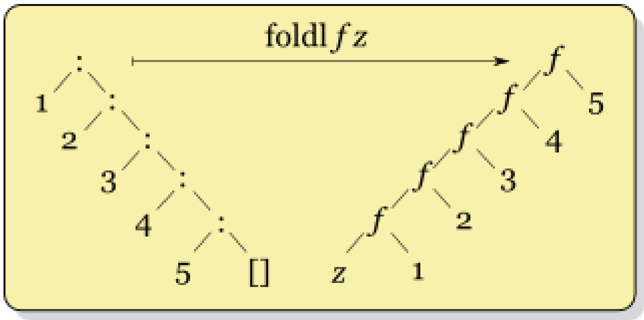
We define a function `foldr` that takes in a function `f` which takes two arguments, and a value `z`. `foldr` essentially replaces the `Link` constructor with `f`, and the empty list with `z`. It then evaluates the expression and returns the result. This is equivalent to:

```
f(1, f(2, f(3, f(4, f(5, z)))))
```

We call this operation a right fold.

Similarly we can define a left fold `foldl` that folds a list starting from the beginning, such that the function `f` will be applied this way:

```
f(f(f(f(f(z, 1), 2), 3), 4), 5)
```



Also notice that a left fold is equivalent to Python's `reduce` with a starting value.

Q11: Fold Left

Write the left fold function by filling in the blanks.

```
def foldl(link, fn, z):
    """ Left fold
    >>> lst = Link(3, Link(2, Link(1)))
    >>> foldl(lst, sub, 0) # (((0 - 3) - 2) - 1)
    -6
    >>> foldl(lst, add, 0) # (((0 + 3) + 2) + 1)
    6
    >>> foldl(lst, mul, 1) # (((1 * 3) * 2) * 1)
    6
    """
    if link is Link.empty:
        return z
    return foldl(link.rest, fn, fn(z, link.first))
```

Use Ok to test your code:

```
python3 ok -q foldl
```

Q12: Fold Right

Now write the right fold function.

```
def foldr(link, fn, z):
    """ Right fold
    >>> lst = Link(3, Link(2, Link(1)))
    >>> foldr(lst, sub, 0) # (3 - (2 - (1 - 0)))
    2
    >>> foldr(lst, add, 0) # (3 + (2 + (1 + 0)))
    6
    >>> foldr(lst, mul, 1) # (3 * (2 * (1 * 1)))
    6
    """
    if link is Link.empty:
        return z
    return fn(link.first, foldr(link.rest, fn, z))
```

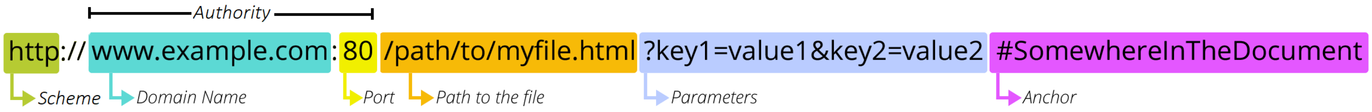
Use Ok to test your code:

```
python3 ok -q foldr
```

Regex

Q13: Basic URL Validation

In this problem, we will write a regular expression which matches a URL. URLs look like the following:


Authority
http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument
Scheme
Domain Name
Port
Path to the file
Parameters
Anchor

For example, in the link `https://cs61a.org/resources/#regular-expressions`, we would have:

- Scheme: `https`
- Domain Name: `cs61a.org`
- Path to the file: `/resources/`
- Anchor: `#regular-expressions`

The port and parameters are not present in this example and you will not be required to match them for this problem.

You can reference this documentation from MDN (https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL) if you're curious about the various parts of a URL.

For this problem, a valid domain name consists of any sequence of letters, numbers, dashes, and periods. For a URL to be "valid," it must contain a valid domain name and will optionally have a scheme, path, and anchor.

A valid scheme will either be `http` or `https`.

Valid paths start with a slash and then must be a valid path to a file or directory. This means they should match something like `/composingprograms.html` or `path/to/file` but not `/composing.programs.html/`.

A valid anchor starts with `#`. While they are more complicated, for this problem assume that valid anchors will then be followed by letters, numbers, hyphens, or underscores.

> **Hint 1**: You can use `\` to escape special characters in regex.

>**Hint 2**: The provided code already handles making the scheme, path, and anchor optional by using non-capturing groups.

```
def match_url(text):
    """
    >>> match_url("https://cs61a.org/resources/#regular-expressions")
    True
    >>> match_url("https://pythontutor.com/composingprograms.html")
    True
    >>> match_url("https://pythontutor.com/should/not.match.this")
    False
    >>> match_url("https://link.com/nor.this/")
    False
    >>> match_url("http://insecure.net")
    True
    >>> match_url("htp://domain.org")
    False
    """
    scheme = r"https?:\/\/"
    domain = r"\w[\w.]+\w"
    path = r"\/(?:[\w\/]+)(?:\.\w+)?"
    anchor = r"#[\w-]+"
    return bool(re.match(rf"^(?:{scheme})?{domain}(?:{path})?(?:{anchor})?$", text))
```

Use Ok to test your code:

```
python3 ok -q match_url
```

BNF

Q14: Simple CSV

CSV, which stands for "Comma Separated Values," is a file format to store columnar information. We will write a BNF grammar for a small subset of CSV, which we will call SimpleCSV.

Create a grammar that reads SimpleCSV, where a file contains rows of words separated by commas. Words are characters `[a-zA-Z]` (and may be blank!) Spaces are not allowed in the file.

Here is an example of a 2-line SimpleCSV file:

```
first,second,third
fourth,fifth,sixth,,eighth
```

We should parse out the following as a result:

```
start
  lines
    line
      word  first
      word  second
      word  third
    newline
    line
      word  fourth
      word  fifth
      word  sixth
      word
      word  eighth
```

```
lines: (line newline)* line newline?

line: (word ",")* word

word: WORD?

newline: "\n"

%import common.WORD
```

Use Ok to test your code:

```
python3 ok -q simple_csv
```