INSTRUCTIONS

- You have 1 hour to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one 8.5” × 11” cheat sheet of your own creation.
- Mark your answers on the exam itself. We will not grade answers written on scratch paper.

<table>
<thead>
<tr>
<th>Last name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>First name</td>
<td></td>
</tr>
<tr>
<td>Student ID number</td>
<td></td>
</tr>
<tr>
<td>Instructional account (cs61a-_)</td>
<td></td>
</tr>
<tr>
<td>BearFacts email (@berkeley.edu)</td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td></td>
</tr>
<tr>
<td>Name of the person to your left</td>
<td></td>
</tr>
<tr>
<td>Name of the person to your right</td>
<td></td>
</tr>
</tbody>
</table>

All the work on this exam is my own. (please sign)
1. (10 points) On My Way to San Jose
For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. If an error occurs, write “Error”. The first box has been filled in for you. Assume that the Link class has been defined. Assume that you have started python3 and executed the following statements:

```python
class City:
    num = 0
    def __init__(self, name, \
                 pop, people=[]):
        self.name = name
        self.pop = pop
        self.people = list(people)
        self.num += 1

class People:
    def __init__(self, place, name, first=0):
        self.place = place
        self.name = name
        if not first:
            self.friend=People(self.place, \
                               "Friend", 1)
        print(self.place.city)
        self.place.city.people.append(self)

class Place(City):
    def __init__(self, name, city=None):
        self.name = name
        self.city = city
        lnk = self.lnk
        while lnk != Link.empty:
            lnk = lnk.rest
        lnk = self

sanjose = City("San Jose", 1)
techmuseum = Place("Tech Museum", sanjose)
steve, bob = People(techmuseum, "Steve"), People(Place("Library", sanjose), "bob")
```

```python
>> sanjose.city
San Jose
>> techmuseum.people
[Steve, Bob]
>> steve.name
Steve
>> bob.name
Bob
>> print(steve.name + " is at " + techmuseum.name)
Steve is at Tech Museum
```
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len(Place.lnk)</code></td>
<td><code>san_jose.goto = People.goto</code></td>
</tr>
<tr>
<td></td>
<td><code>san_jose.goto(tech_museum)</code></td>
</tr>
<tr>
<td><code>bob.goto(tech_museum)</code></td>
<td><code>san_jose.goto = steve.goto</code></td>
</tr>
<tr>
<td></td>
<td><code>san_jose.goto(tech_museum)</code></td>
</tr>
<tr>
<td><code>print(bob.goto(san_jose))</code></td>
<td><code>berkeley = City(&quot;Berkeley&quot;, 2, \</code></td>
</tr>
<tr>
<td></td>
<td><code>[steve, bob])</code></td>
</tr>
<tr>
<td></td>
<td><code>City.num</code></td>
</tr>
<tr>
<td><code>People.__init__(san_jose, \</code></td>
<td><code>berkeley.people[0] == \</code></td>
</tr>
<tr>
<td><code>san_jose, &quot;Yali’s&quot;)</code></td>
<td><code>san_jose.people[1]</code></td>
</tr>
<tr>
<td></td>
<td><code>[i.name for i in berkeley.people]</code></td>
</tr>
<tr>
<td><code>san_jose.name</code></td>
<td></td>
</tr>
<tr>
<td><code>san_jose.city = san_jose</code></td>
<td></td>
</tr>
<tr>
<td><code>People.__init__(san_jose, \</code></td>
<td></td>
</tr>
<tr>
<td><code>san_jose, &quot;Yali’s&quot;)</code></td>
<td></td>
</tr>
<tr>
<td><code>san_jose.name</code></td>
<td></td>
</tr>
</tbody>
</table>
2. (10 points)  Aaaaaaa

Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs. You may not need to use all of the spaces or frames.

A complete answer will:

- Add all missing names and parent annotations to all frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```python
1  w, x = lambda s: x, lambda: x
2  def a(x):
3      def a():
4          nonlocal a
5          b()
6          a = w
7
8  def b():
9      nonlocal a
10     a = [x]
11     for i in range(2):
12         a.append(a[i-1]())
13     a()
14  return a
15  x = a(x)
16  w = x(a)
```
3. (10 points) Scheme-ing Merge
Given two sorted lists, lst1 and lst2, return a list that sorts both in ascending order. Break ties in any way you wish.

```
(define (merge lst1 lst2)
  (cond ((______________________) ________________________________)
        ((______________________) ________________________________)
        ((______________________) ________________________________)
        (else (______________________________))))
```

4. (10 points) Scheme-ing to Find a Path
Here is the BinTree class provided for your reference:

```python
class BinTree:
    empty = ()
    def __init__(self, label, left=empty, right=empty):
        self.label = label
        self.left = left
        self.right = right
```

Given a binary search tree and an entry, return the path in order to reach the entry from the root in the form of a list.

```python
def pathfinder(bst, entry):
    ""
    >>> bintree = BinTree(4, BinTree(2, BinTree(1)), BinTree(5))
    >>> pathfinder(bst, 2)
    [4, 2]
    >>> pathfinder(bst, 1)
    [4, 2, 1]
    ""
    if ____________________________:
        ________________________________
    elif ________________________________:
        ________________________________
    elif ________________________________:
        return ________________________________
    else:
        return ________________________________
```
5. (10 points) Homework Party: The SQL

You are a veteran at RuneSQL, a popular RPG (role-playing game) where you hone your skills to become the best player in the database! However, you are a little short on SUPER DUPER EPIC RARE 61A homework party hats. Other players (a.k.a. n00bs) are fortunately predictable. Through your many years of being a crafty RuneSQL economist, you have taken note of the trends in hat_prices. The following chart shows the price per unit (in millions of RuneSQL coins) and quantity for a batch offer of party hats at a certain time (in minutes).

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>0.3</td>
<td>10</td>
</tr>
<tr>
<td>60</td>
<td>0.75</td>
<td>40</td>
</tr>
<tr>
<td>90</td>
<td>0.7</td>
<td>25</td>
</tr>
<tr>
<td>120</td>
<td>1.3</td>
<td>25</td>
</tr>
<tr>
<td>150</td>
<td>1.25</td>
<td>30</td>
</tr>
<tr>
<td>180</td>
<td>0.4</td>
<td>5</td>
</tr>
<tr>
<td>210</td>
<td>0.45</td>
<td>10</td>
</tr>
</tbody>
</table>

There's a catch! You will have to wait 1 hour after buying a single batch of hats or n00bs will get suspicious and market prices will change. Write a SQL select statement to show you the path to the maximum number of hats you can buy for 50 million coins, your current budget.

-- Expected result:
-- 0, 60, 210|70

WITH paths(path, prev_time, units, money) as (  
SELECT _______________________
FROM hat_prices UNION  
SELECT _______________________
FROM hat_prices, paths  
WHERE money >= 0 and time - prev_time > 30
)
SELECT _______________________ FROM _______________________;