

### INSTRUCTIONS

This is your exam. Complete it either at [exam.cs61a.org](http://exam.cs61a.org) or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address <EMAILADDRESS>. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- You must choose either this option
- Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- You could select this choice.
- You could select this one too!

**You may start your exam now. Your exam is due at <DEADLINE> Pacific Time.** Go to the next page to begin.

## Preliminaries

You can complete and submit these questions before the exam starts.

### 0.0.1 Basic Directions

- You have 3 hours, 180 minutes to complete the exam.
- You **must not** collaborate with anyone inside or outside of CS88.
- You may use the internet, the CS88 site and all it's resources,
- *However*, you **must not** directly search for a question or post questions online.
- You may search for generic Python concepts.
- You may use your Terminal and Python Tutor.
  - However, these are more strict about syntax! The exam is designed to be completed *without* these tools, and using them may take up some time. Be mindful of how long you spend on each question.
- You should have received an email with a link for your Zoom meeting, and ideally it will show up in the Zoom app when signed in. If you can, please use that.
- At this point you should have started your Zoom / screen recording. If something happens during the exam, focus on the exam!
- Do not spend more than a few minutes dealing with proctoring.
- Your task is to show us how much you've learned, not to mess with technology.

**Take a deep breath...**

*Good luck, and good skill!*

(a) What is your full name?

(b) What is your student ID number?

(c) What is your Berkeley email address?

## 1. Reference Links

### 0.0.2 References

These are references to use during the exam. When you see a **Link** or **Tree**, you should use these as the class definitions.

**Clarifications Document:** <https://docs.google.com/document/d/16-L1PDa63Jn-FZ63CsPJbIGLeDkYV9gng4IISNwFMD>

**Reference Sheet:** <https://drive.google.com/file/d/1bTRKCwtGSo4MMd42bv7YGYgyEyDUOnBq/view?usp=sharing>

**Tree Class:**

```
class Tree:
    def __init__(self, entry, branches=()):
        self.entry = entry
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

    def __repr__(self):
        if self.branches:
            branches_str = ', ' + repr(self.branches)
        else:
            branches_str = ''
        return 'Tree({0}{1})'.format(self.entry, branches_str)

    def __str__(self):
        def print_tree(t, indent=0):
            tree_str = '  ' * indent + str(t.entry) + "\n"
            for b in t.branches:
                tree_str += print_tree(b, indent + 1)
            return tree_str
        return print_tree(self).rstrip()

    def is_leaf(self):
        return not self.branches
```

**Linked List Class**

```
class Link:
    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

    def __repr__(self):
        if self.rest is not Link.empty:
            rest_str = ', ' + repr(self.rest)
        else:
            rest_str = ''
        return 'Link({0}{1})'.format(repr(self.first), rest_str)
```

(There is nothing to submit for question 1.)

**2. (2.0 points) Conceptual Questions**

- (a) (1.0 pt) Explain how the Place Class in the Ants Vs. SomeBees Project models a Linked List data structure.



(b) (1.0 pt) Suppose we have a Restaurant class with the following attributes and methods:

- `__init__(self, restaurant_name, menu, available_tables)`: creates instance attributes `restaurant_name`, `menu`, `available_tables`
- `add_item(self, name)`: adds a new menu item to menu
- `cook(self, order)`: prepares a single order for a customer
- `clean_table(self)`: `available_tables` increments by 1
- `update_price(self, menu_item)`: updates the price of an item on the menu

Using your knowledge of OOP and inheritance, create an example subclass that would inherit the `Restaurant` class and explain how you would make use of inheritance. No code needs to be written for this problem, a brief (2-4 sentences) description and justification is sufficient.

(c) (1.0 pt) Which data structure would you use to store a family's heritage and why?

A large empty rectangular box with a thin black border, intended for the student to write their answer to the question.

**3. (7.0 points) What Made Python Print That?**

Given the following lines of code and respective output, fill in the blanks to produce the desired result.

**(a) Mystery**

```
>>> def mystery(l, n, f):
    if l is Link.empty:
        return n == 0
    return f(n, l.first) and mystery(l.rest, n // 10, f)

>>> l = -----
>>> n = -----
>>> l.first != l.rest.first and mystery(l, n, lambda x, y: x % 10 == y)
True
```

Fill in the blanks for the values of `l` and `n` such that the final expression evaluates to `True`.

**i. (3.0 pt) Blank (l)****ii. (3.0 pt) Blank (n)**

**(b) Music Album**

For this problem, we will be working with the `MusicAlbum` class, where each `MusicAlbum` instance has two attributes, the `album_name` and a `votes_per_song` dictionary that has songs in the album as *keys* and the number of votes per song as *values*.

Though the following `MusicAlbum` class is syntactically correct, there is a bug with the logic in the `most_popular_song_in_album` method. The method is supposed to find the album with the most votes, and though it sometimes works correctly, the logic bug makes it incorrect for some sequence of method calls.

```
class MusicAlbum:
    def __init__(self, album_name, songs):
        self.album_name = album_name
        self.votes_per_song = {}
        for song in songs:
            self.votes_per_song[song] = 0

    def up_vote(self, song):
        assert song in self.votes_per_song
        self.votes_per_song[song] += 1

    def down_vote(self, song):
        assert song in self.votes_per_song
        self.votes_per_song[song] -= 1

    def most_popular_song_in_album(self):
        most_popular_song = ""
        most_popular_votes = 0
        for song in self.votes_per_song:
            if self.votes_per_song[song] >= most_popular_votes:
                most_popular_song = song
                most_popular_votes = self.votes_per_song[song]
        return most_popular_song
```



- i. (2.0 pt) Briefly describe what the issue in the `most_popular_song_in_album` method is (no more than 30 words). Then come up with a sequence of method calls that would expose the bug in the current code (no more than 5 lines).

- ii. (2.0 pt) What is the runtime of the `most_popular_song_in_album` method in terms of  $N$ , which is the number of songs in the `self.votes_per_song` dictionary?

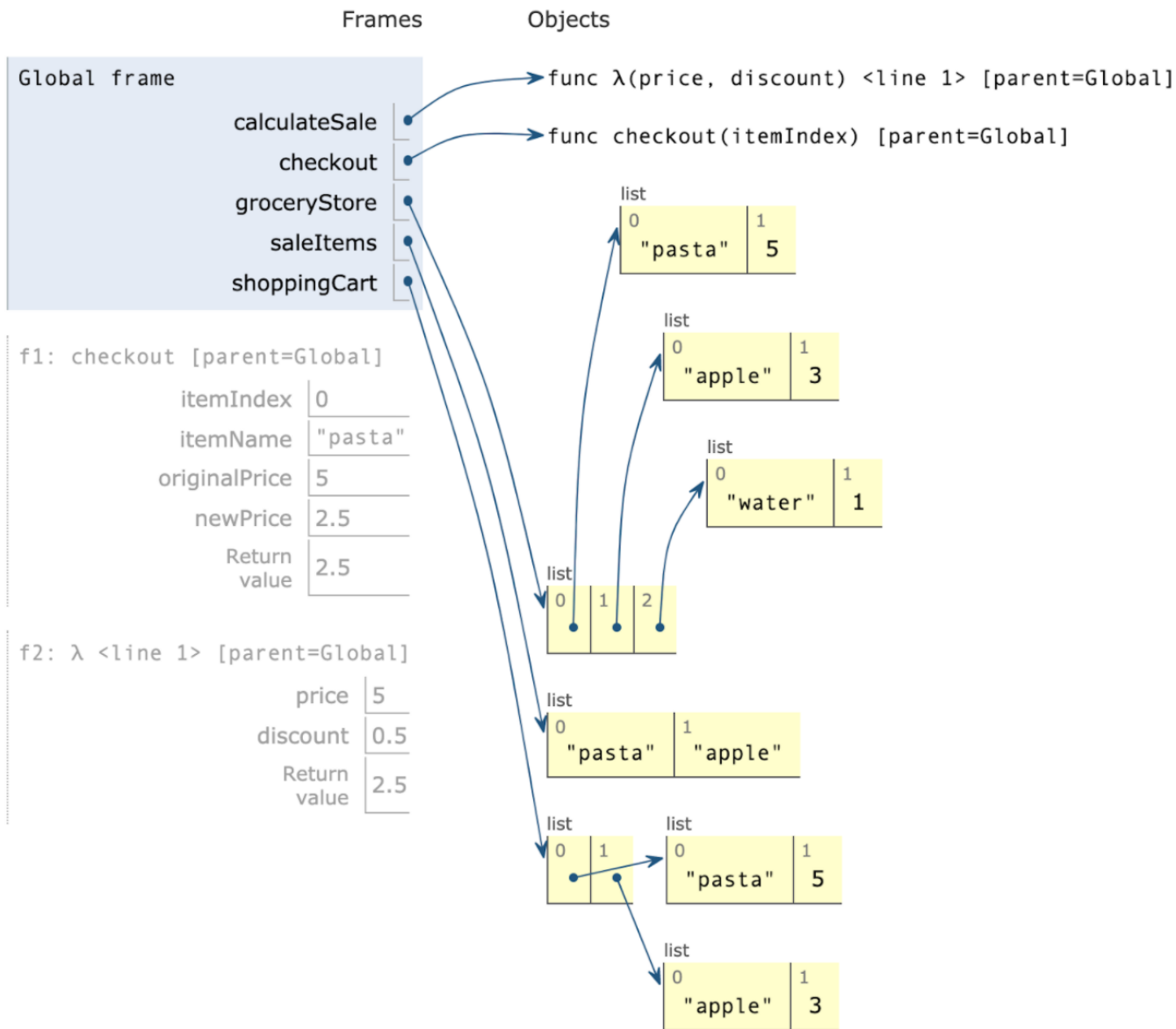
- $O(1)$
- $O(\log(N))$
- $O(N)$
- $O(N^2)$
- $O(2^N)$

#### 4. (9.0 points) Let's Go Grocery Shopping

**Note: Please use: <http://tutor.cs61a.org/> to check your environment diagram if needed!**

In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram. You may want to fill in the blanks in a different order; feel free to answer the questions in whatever order works for you.

The following environment diagram was generated by a program to completion:



(Click to Open Image)

`calculateSale = _____`  
(a)

```
def checkout(itemIndex):
    itemName = _____
    (b)
    if itemName in saleItems:
        originalPrice = groceryStore[itemIndex][1]
```

```

        newPrice = calculateSale(originalPrice, 0.5)
        return newPrice
    else:
        return groceryStore[itemIndex][1]

groceryStore = [["pasta", 5], ["apple", 3], ["water", 1]]
saleItems = ["pasta", "apple"]
shoppingCart = -----
                (c)
checkout(0)

```

**Hint:** Try solving blank c first before blanks a and b!

(a) (3.0 pt) Which of these could fill in blank (a)? *Select one answer from below.*

- lambda price, discount: originalPrice \* discount
- lambda price, discount : price / discount
- lambda price, discount : price + discount
- lambda price, discount : price \* discount

(b) (3.0 pt) Which of these could fill in blank (b)? *Check all that apply.*

- groceryStore["pasta"][0]
- groceryStore[itemIndex][0]
- groceryStore[0][0]
- groceryStore[-3][0]

(c) (3.0 pt) Which of these could fill in blank (c)? *Select one answer from below.*

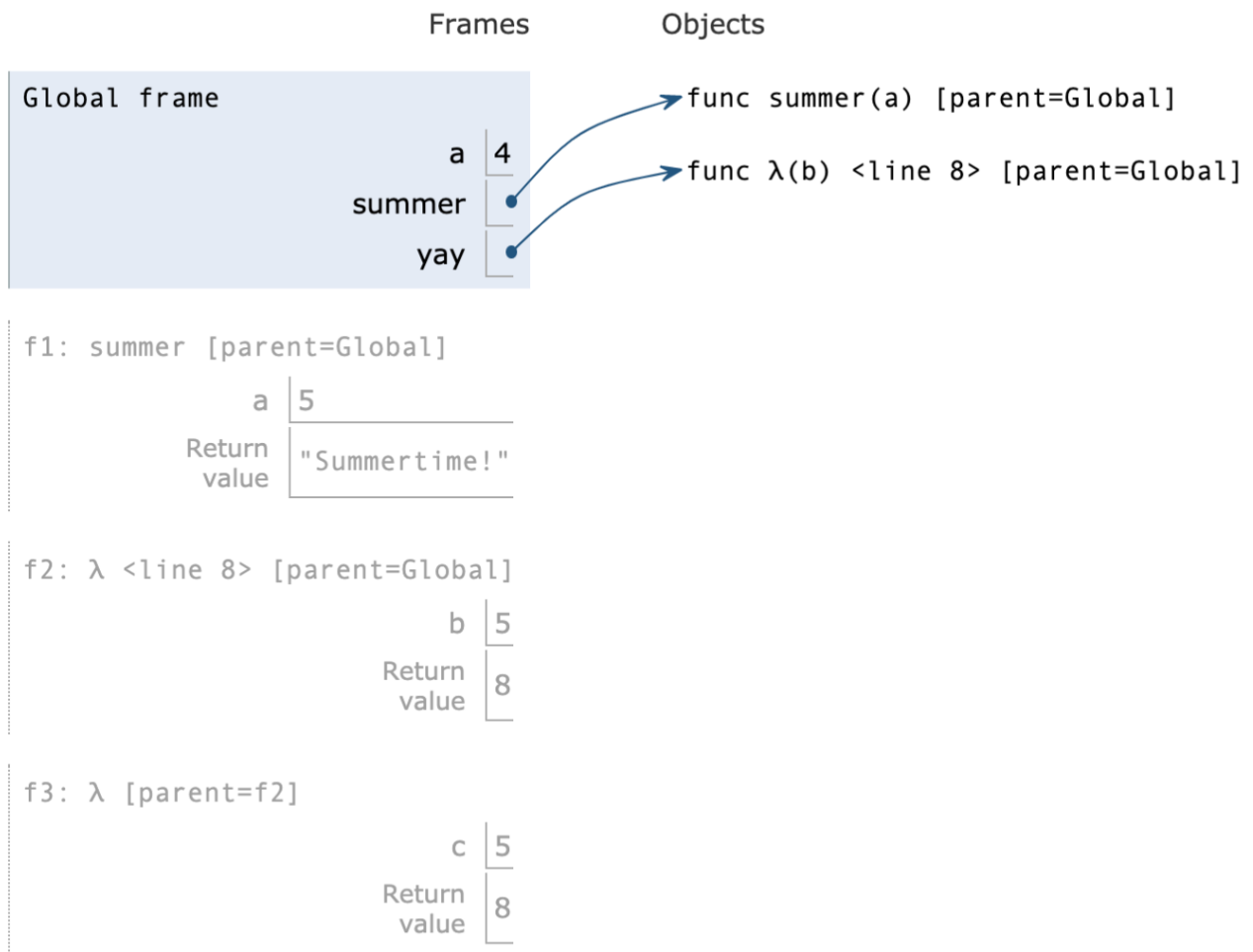
- [groceryStore[0]] + [groceryStore[1]]
- [groceryStore[0][:]] + [groceryStore[1][:]]
- groceryStore[0][:].append(groceryStore[1][:])
- ["pasta", 5, "apple", 3]

## 5. (9.0 points) Summer is finally here!

Note: Please use: <http://tutor.cs61a.org/> to check your environment diagram if needed!

In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram. You may want to fill in the blanks in a different order; feel free to answer the questions in whatever order works for you.

The following environment diagram was generated by a program to completion:



(Click to Open Image)

```

a = 4
def summer(a):
    if _____:
        (a)
        return "Summertime!"
    else:
        return "Not yet :("

yay = _____
      (b)

summer(_____)
      (c)

```

(a) (3.0 pt) Which of these could fill in blank (a)? *Check all that apply.*

`yay(a) == 8`

`yay(a) >= 7`

`yay(a) < 10`

`yay(a) == 7`

(b) (3.0 pt) Which of these could fill in blank (b)? *Select one answer from below.*

`lambda b: (lambda c: a*2) (b)`

`lambda b: (lambda c : 8)`

`lambda b: (lambda c: 8) (a)`

`lambda b: (lambda c: b*2) (5)`

(c) (3.0 pt) Fill in blank (c) and please write out the entire line of code below.

**6. (7.0 points) Time's up, let's do this!**

Create a class class `Clock` which keeps track of hours and minutes. The constructor takes in the `hour` and `minute` that the clock should be set to initially. It also has the following two functions:

- `advance(self)`:
  - This function should advance the minutes by 1 minute
  - If the minute is greater than or equal to 60 then minutes should be set to 0 and hour should be advanced by 1 hour
  - If hour is greater than 12 then hour should be set to 0
- `sync(self)`:
  - This function should synchronize all clocks ever created to have the same hour and minute to the instance that this function is called on (see doctests for an example).

Also fill in the `FastClock` class which inherits from `Clock`. The `advance` method of the `FastClock` should advance the minutes twice as much as a `Clock`. You cannot use more than the lines provided, but feel free to use leave some lines blank.

```
class Clock:
    """
    >>> regular = Clock(12, 59)
    >>> fast = FastClock(12, 59) #Both clocks are initialized to 12 hours and 59 minutes
    >>> print(regular.hour, regular.minute)
    12 59
    >>> print(fast.hour, fast.minute)
    12 59
    >>> regular.advance()
    >>> fast.advance() #Fast clocks advance twice as fast
    >>> print(regular.hour, regular.minute)
    1 0
    >>> print(fast.hour, fast.minute)
    1 1
    >>> another = Clock(6, 0) #initialize another clock to 6 hours and 0 minutes
    >>> regular.sync() #Each clock is synchronized to regular's time.
    >>> print(regular.hour, regular.minute)
    1 0
    >>> print(another.hour, another.minute)
    1 0
    >>> print(fast.hour, fast.minute)
    1 0
    """
    -----
    -----

    def __init__(self, hour, minutes):
        #make sure that hour and minutes are in range
        assert hour <= 12 and hour > 0 and minutes < 60 and minutes > 0

        -----
        -----
        -----
        -----

    def advance(self):
        -----
        -----
        -----
        -----
```

```
-----  
-----  
-----  
-----  
def sync(self):  
-----  
-----  
-----  
-----
```

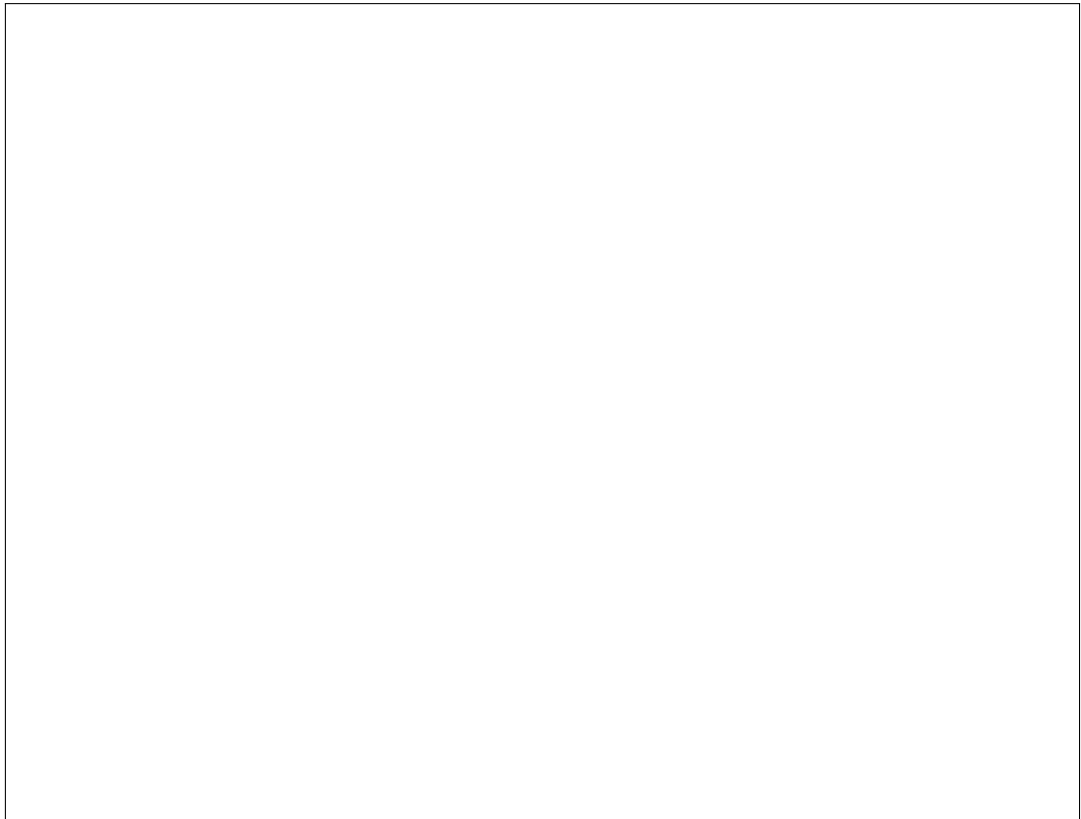
```
class FastClock(Clock):  
-----  
-----  
  
def advance(self):  
-----  
-----  
-----
```

(a) (3.0 pt) Fill in the class attributes for the Clock class.

```
class Clock:  
-----  
-----
```

(b) (3.0 pt) Complete the init function for the Clock class.

```
def __init__(self, hour, minutes):  
    # make sure that hour and minutes are in range  
    assert hour <= 12 and hour > 0 and minutes < 60 and minutes > 0  
    -----  
    -----  
    -----  
    -----
```





(c) (3.0 pt) Complete the `advance` function for the `Clock` class.

```
def advance(self):
```

```
-----  
-----  
-----  
-----  
-----  
-----  
-----  
-----
```

(d) (3.0 pt) Complete the `sync` function for the `Clock` class.

```
def sync(self):
```

```
-----  
-----  
-----  
-----
```



(e) (3.0 pt) Fill in the class attributes for the FastClock class.

```
class FastClock(Clock):
```

```
    -----  
    -----
```



(f) (3.0 pt) Complete the `advance` function for the `FastClock` class.

```
def advance(self):
```

```
-----  
-----  
-----
```



**7. (7.0 points) Coupons!**

Fill in the blanks below to create a coupon tracker that has the functionality in the doctest below. Understanding the doctests will be especially important for this problem.

`coupon_tracker` returns the function `add_coupon` which adds `num_copies` copies of the coupon represented by the string `code` to the dictionary `count`.

Coupons can be added by calling `add_coupon` but when the string "finish" is passed in as `code`, `add_coupon` returns another function that allows coupons to be acquired.

Coupons can be acquired by calling `acquire_coupon` but to successfully acquire a coupon, it must be in the dictionary and have a positive number of copies. Each time a coupon is successfully acquired its number of copies is decreased by 1.

```
def coupon_tracker():
    """
    >>> coupon_adder = coupon_tracker()
    >>> coupon_adder("X", 1)
    >>> coupon_adder("Y", 1)
    >>> coupon_adder("X", 1) # now there are 2 coupons with code "X"
    >>> acquire = coupon_adder("finish")# Returns a function
    >>> acquire("Z") # "Z" is not a coupon code
    Failure.
    >>> acquire("X") # After this, 1 coupon with code "X" remain
    Success!
    >>> acquire("Y")
    Success!
    >>> acquire("X") # After this, 0 coupons with code "X" remain
    Success!
    >>> acquire("X")
    Failure.
    """
    count = {}
    def acquire_coupon(code):
        if -----:
            -----
            print("Success!")
        else:
            print("Failure.")

    def add_coupon(code, num_copies=0):
        if code == "finish":
            -----
        elif code in count:
            -----
        else:
            -----
    return add_coupon
```


(a) (7.0 pt) Fill in the blanks to complete the `acquire_coupon` function.

```
def acquire_coupon(code):  
    if -----:  
        -----  
        print("Success!")  
    else:  
        print("Failure.")
```



(b) (7.0 pt) Fill in the blanks to complete the `add_coupon` function.

```
def add_coupon(code, num_copies=0):  
    if code == "finish":  
        -----  
    elif code in count:  
        -----  
    else:  
        -----  
    return add_coupon
```



**8. (10.0 points) List of Links**

Your friend is trying to write a function `combine` which takes in a list of linked lists. It concatenates each of the linked lists in sequential order and returns the result. Mutating the linked lists is allowed, and the list that is passed in is guaranteed to have at least one linked list. However, their code is buggy.

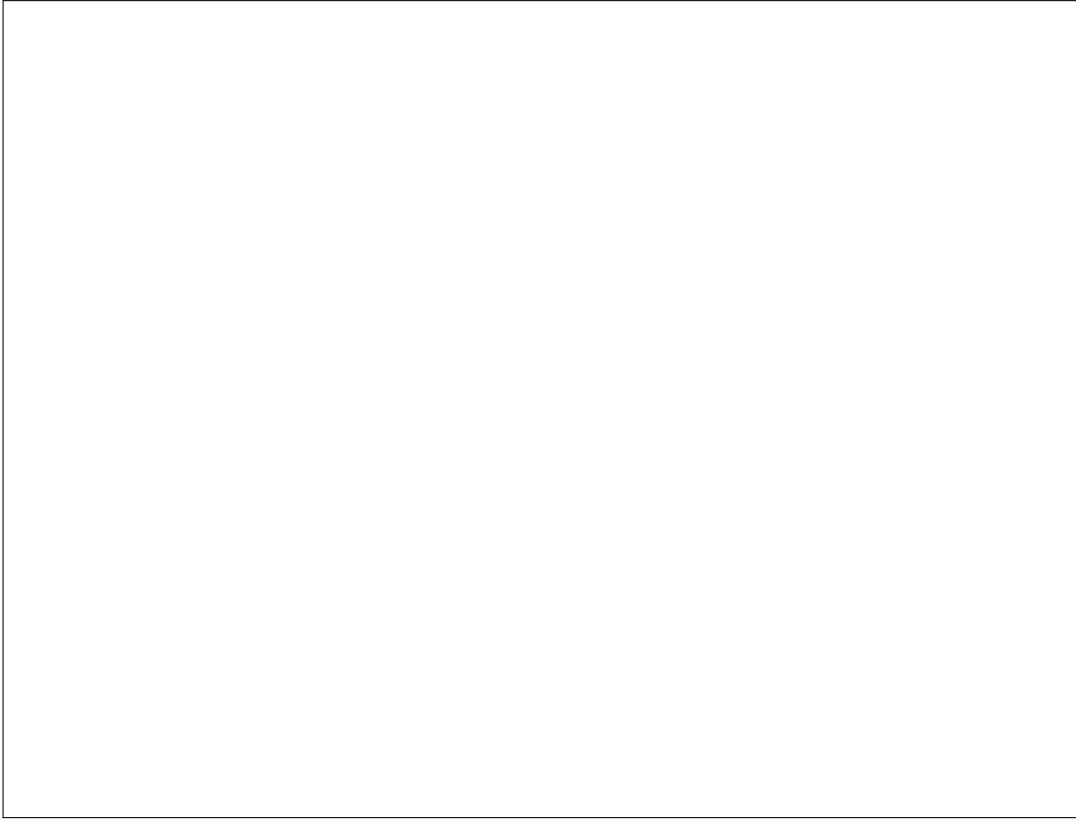
Describe 3 bugs with the code and the way to fix each of the bugs. After fixing each of the bugs, the code should work as intended, and all of the doctests should output the expected output. You may only provide 3 bugs, if you provide more than 3 bugs then only the first 3 will be considered.

**The doctests display the correct output for the function.**

```
(a) def combine(lst):
    """
    >>> lst1 = [Link(1), Link(2), Link(3)]
    >>> combine(lst1)
    Link(1, Link(2, Link(3)))
    >>> lst2 = [Link(1,Link(4, Link(5))), Link(2), Link(3,Link(2))]
    >>> combine(lst2)
    Link(1, Link(4, Link(5, Link(2, Link(3, Link(2))))))
    >>> lst3 = [Link(1)]
    >>> combine(lst3)
    Link(1)
    """
    1.     for i in range(len(lst)):
    2.         curr = lst[i]
    3.         while curr.rest:
    4.             curr = curr.rest
    5.         curr.rest = lst[i + 1].first
    6.     return curr
```



- i. (6.0 pt) Describe the 3 bugs you found in the `combine` function and explain how to fix each bug.



ii. (6.0 pt) Write a working solution to the combine function.

```
def combine(lst):
```

```
-----  
-----  
-----  
-----  
-----  
-----  
-----  
-----
```

- (b) i. (4.0 pt) Write a working function for `product_slice`.

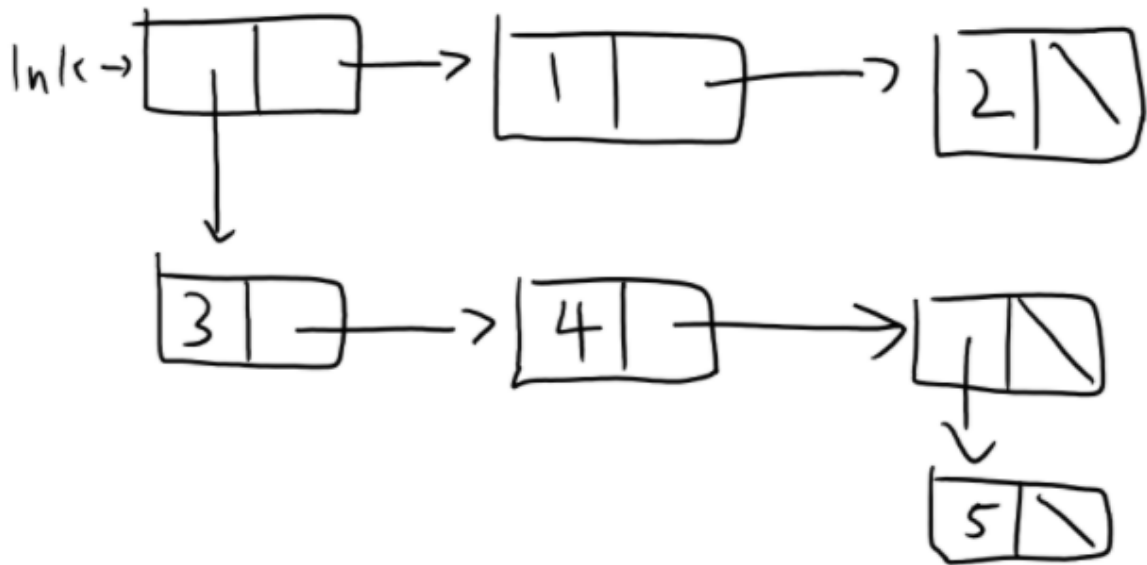
A large empty rectangular box with a thin black border, intended for the student to write their code for the `product_slice` function.

**9. (10.0 points) Surprise Holiday**

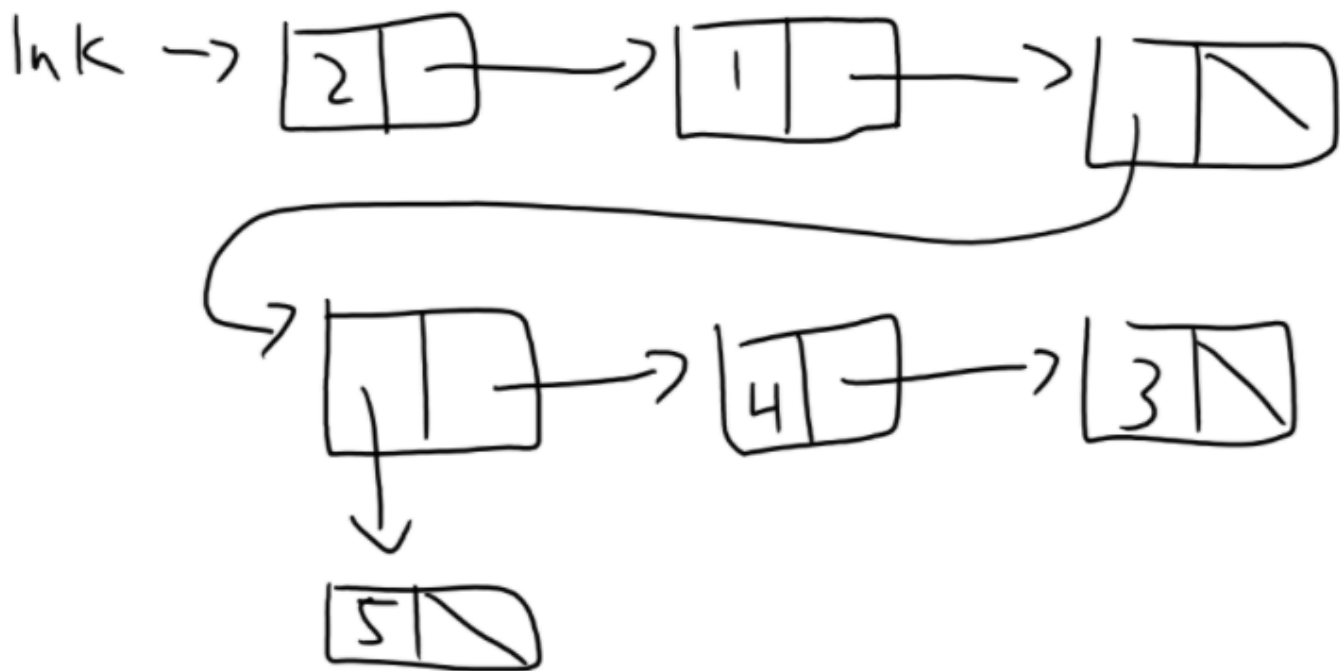
Paul is just about to submit his test linked list inputs for his cs88 assignment a couple hours before the due date. He checks the calendar one last time, and gasps in horror - he forgot it was opposite day. According to the laws of the universe, he must now submit all his linked lists BUT REVERSED. In this problem you will help Paul write a function so that he can create new linked list inputs to submit in time.

Paul's Linked Lists are not simple linked lists, they are nested, meaning that each linked list can have a first value that is another linked list (and those can have linked lists as their first value and so on). It is your responsibility to write a function `nested_reverse` that reverses nested linked list, and all nested lists within the list, and returns a new list. Below is a pictorial example.

Hint: `isInstance(Link.first, Link)` checks whether or not `Link.first` is of type `Link` (meaning that it is a linked list). We gave this part in the skeleton as a hint for the function.



After :



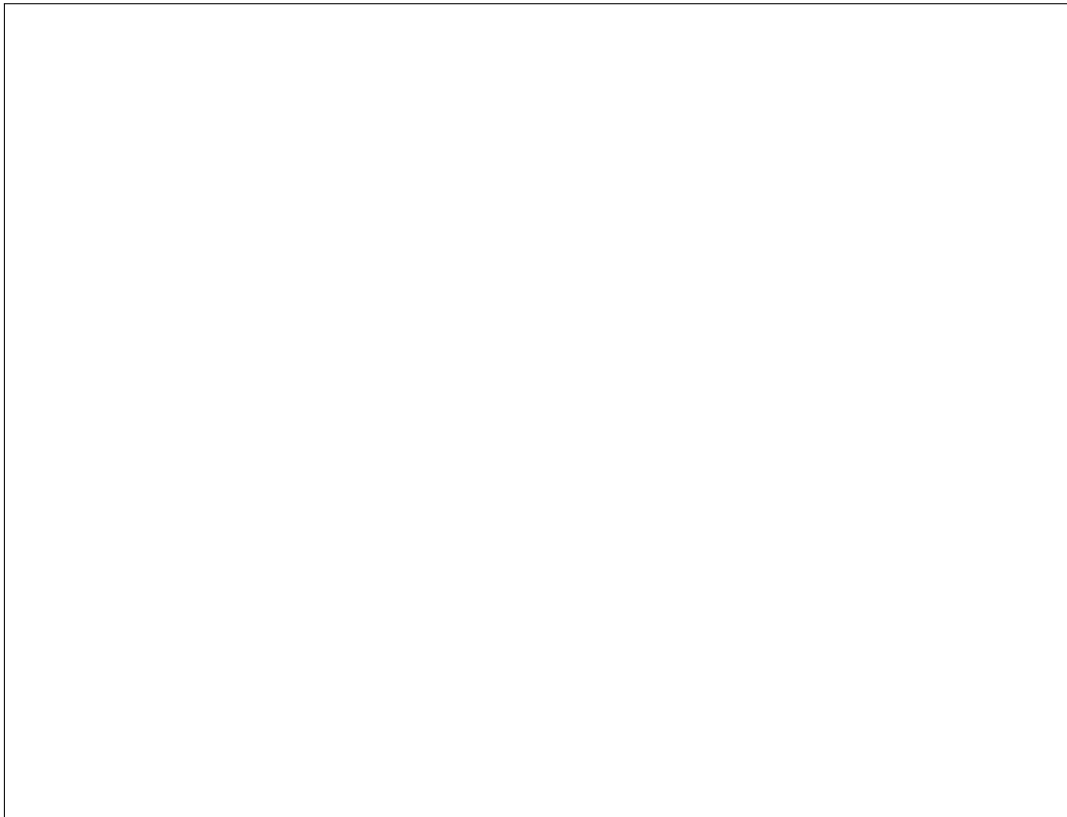
(Click to Open Image)

```
def nested_reverse(link):
    >>> nested_reverse(Link(Link(3, Link(4, Link(Link(5))))), Link(1, Link(2)))
    Link(2, Link(1, Link(Link(Link(Link(5), Link(4, Link(3)))))) #above example
    >>> nested_reverse(Link(Link(1)))
    >>> Link(Link(1)) #Nothing to reverse

new = Link.empty
```

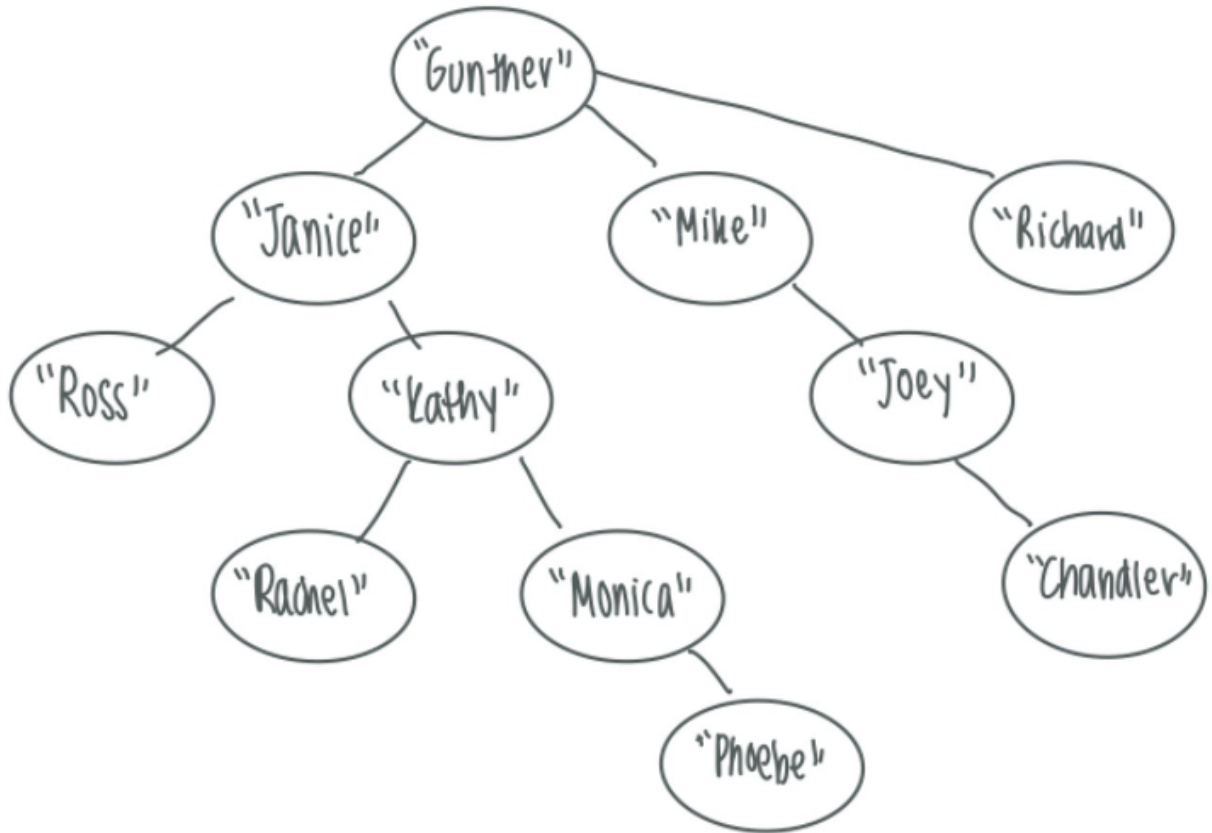
```
while -----:  
    if isinstance(link.first, Link):  
        -----  
    else:  
        -----  
        -----  
return -----
```

(a) (4.0 pt) Fill in the blanks to complete the `nested_reverse` function.



10. (10.0 points) Working in the IndusTREE

You have been recently hired by Company Z to help the company's recruiting team! Company Z's organizational structure can be represented as a tree (see the image below) with each employee as a branch of their manager. Note that it is possible for a manager to also be an employee, e.g. Kathy reports to Janice, and Rachel and Monica report to Kathy! By this representation, the president is the root node!



(Click to Open Image)

- (a) (4.0 pt) For your first task, you are asked to consider how many openings managers have on their teams for potential new hires. A manager is anyone that has at least one person reporting to them. At company Z, no manager should have more than  $k$  employees immediately reporting to them.

Given a tree  $t$  representing company Z and the number  $k$  representing the max number of employees reporting to one manager, return the number of openings managers have on their teams for new hires across the company. You can assume that in the current tree  $t$ , no manager is managing more than  $k$  employees.

```
def num_openings(t, k):
    """
    >>> finance = Tree("Janice", [Tree("Ross"), Tree("Kathy", [Tree("Rachel"),
    ...     Tree("Monica", [Tree("Phoebe")])])])
    >>> marketing = Tree("Mike", [Tree("Joe", [Tree("Chandler")])])
    >>> operations = Tree("Richard")
    >>> t = Tree("Gunther", [finance, marketing, operations])
    >>> num_openings(t, 3)
    8
    """
    if _____:
        return _____
    else:
        _____
        for _____:
            _____
        return _____
```

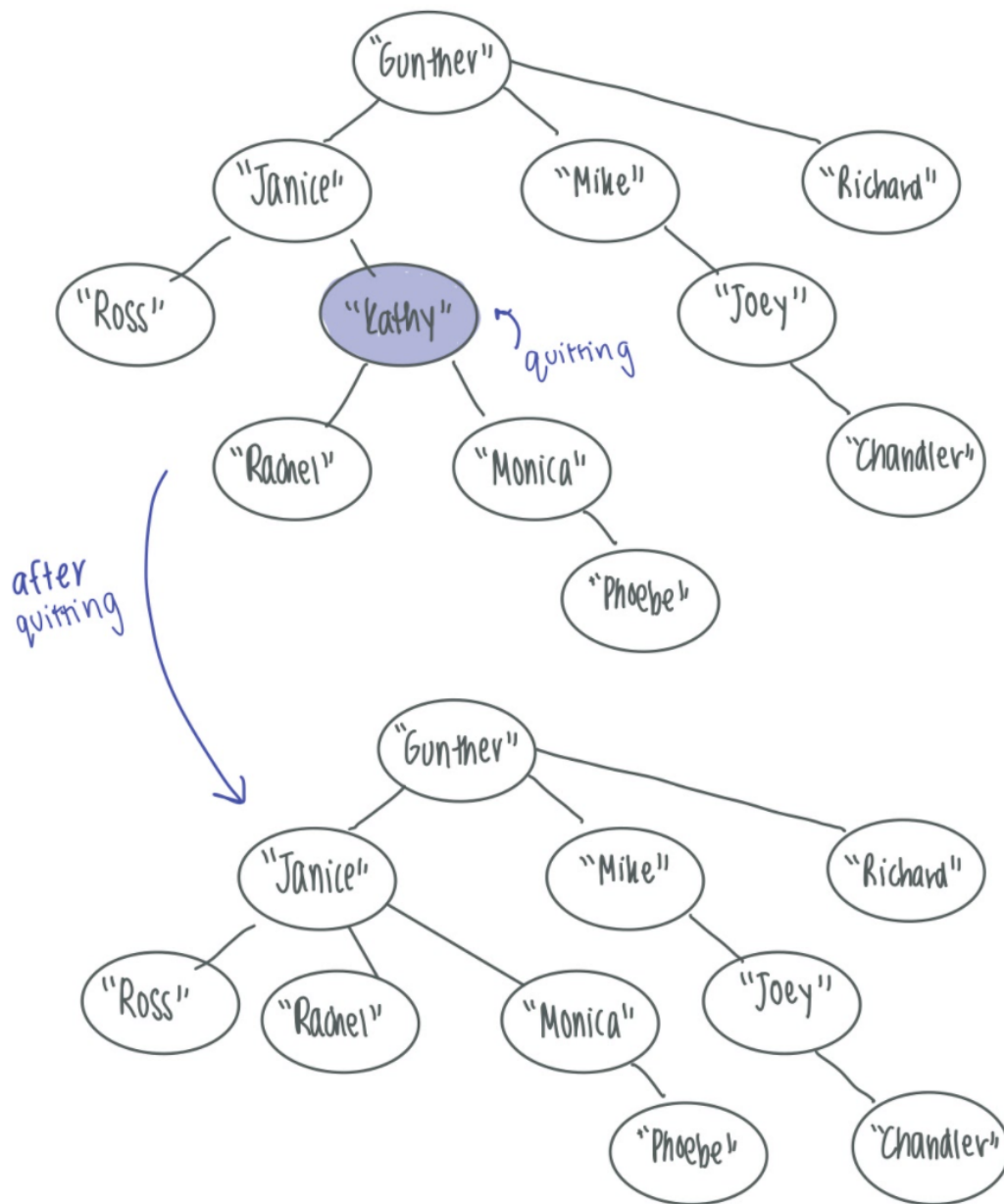
Fill in the blanks to complete the `num_openings` function.



- (b) (4.0 pt) For your second task, you have been told that there is an employee quitting the company because they've found a new job opportunity.

Given a tree  $t$  representing company Z, modify the tree  $t$  to remove the employee that is leaving, moving everyone who previously reported to that employee as now reporting to that employee's previous manager. For the sake of this problem, you can assume that there are no longer any constraints on how many employees can report to a manager.

You can assume that the president (or the root node) will never leave the company.



(Click to Open Image)

```
def quit(t, employee):  
    """  
    >>> finance = Tree("Janice", [Tree("Ross"), Tree("Kathy", [Tree("Rachel"),
```

```
...     Tree("Monica", [Tree("Phoebe")]))]))
>>> marketing = Tree("Mike", [Tree("Joe", [Tree("Chandler")])])
>>> operations = Tree("Richard")
>>> t = Tree("Gunther", [finance, marketing, operations])
>>> quit(t, "Kathy")
>>> print(t)
Gunther
  Janice
    Ross
    Rachel
    Monica
      Phoebe
  Mike
    Joe
      Chandler
  Richard
"""

for -----:
  if -----:
    -----
    -----
    -----

for -----:
  -----
```

Fill in the blanks to complete the quit function.

**11. (10.0 points) Root Path Sums**

The year is 2093 and the world has fallen into a post-apocalyptic nuclear winter. Michael Ball prime, a copy of Michael Ball's consciousness put into a robotic exoskeleton, needs to travel away from its shelter to charge himself in nearby cities.

Michael Ball prime starts at the root of a tree, and travels along branches to each node, where he charges himself according to the value (charge points) at that node. But beware, Michael Ball prime's battery can only hold  $n$  charge points, and will melt down if given anymore.

**Write a procedure to find all paths that sum to exactly  $n$  charge points so that he becomes fully charged. In other words, find all the paths in the tree (starting at the root) where the nodes add up to  $n$ .**

(a) (4.0 pt)

```
def root_path_sums(t, n):
    """
    >>> t = Tree(1, [Tree(2, [Tree(1)]), Tree(3)])
    >>> gen = root_path_sums(t, 4)
    >>> next(gen)
    [1, 2, 1]
    >>> next(gen)
    [1, 3]
    >>> next(gen)
    StopIteration error
    >>> gen2 = root_path_sums(t, 15)
    >>> next(gen2)
    StopIteration error
    """
    if t.label == n:
        yield [t.label]
    else:
        for _____:
            for _____ in root_path_sums(_____, _____):
                cur_path = [t.label] + _____
                yield _____
```

Fill in the blanks to complete the `root_path_sums` function.

**12. (10.0 points) Post Final Dessert!**

```
CREATE TABLE ice_cream AS
  SELECT "vanilla" as flavor, "classic" as category UNION
  SELECT "chocolate", "classic" UNION
  SELECT "strawberry", "fruits" UNION
  SELECT "mango", "fruits" UNION
  SELECT "coffee", "fancy" UNION
  SELECT "mint chocolate chip", "fancy";
```

```
CREATE TABLE staff AS
  SELECT "Vandana" as name, "coffee" as favorite UNION
  SELECT "Shreya", "strawberry" UNION
  SELECT "Sophia", "mango" UNION
  SELECT "Nick", "vanilla" UNION
  SELECT "Lukas", "mango" UNION
  SELECT "Tommy", "mint chocolate chip" UNION
  SELECT "Kevin", "strawberry" UNION
  SELECT "Minnie", "chocolate" UNION
  SELECT "Matt", "vanilla" UNION
  SELECT "Michael", "coffee" UNION
  SELECT "Gerald", "mango";
```

Use the above tables to write queries below.

- (a) (4.0 pt) Write a SELECT query which would output the names of people on staff who like ice cream in the fruits category.

Output:

Shreya

Sophia

Lukas

Kevin

Gerald



**(b) (4.0 pt)**

Code Blocks

- 1. SELECT a.name -----
- 2. GROUP BY ice\_cream\_a.category -----
- 3. HAVING COUNT(a.name) + COUNT(b.name) = 2
- 4. ORDER BY -----
- 5. FROM staff as a, staff as b -----
- 6. WHERE -----

Write a SELECT query using the blocks of code given above to match everyone in the staff table in pairs if they like ice cream in the classic category.

*Not all the blocks may need to be used and they may be incomplete or incorrect. The blocks are also out of order.*

HINT: use the output to determine how to handle duplicates

Output:

```
Nick | Minnie  
Nick | Matt  
Minnie | Matt
```

- (c) (4.0 pt) Write a SELECT query to output how much of each ice cream flavor is needed for everyone to get their favorite flavor in decreasing popularity. If the popularity is tied, break ties alphabetically. Do not include flavors where only one person likes the flavor.

HINT: To order by two columns, you can use a comma to separate the columns ex. ORDER BY col1, col2

Output:

Output:

mango	3
coffee	2
strawberry	2
vanilla	2



**13. (10.0 points) Congratulations!**

Woohoo! You are now officially done with CS88! Thanks so much for all of your hard work this semester. We are incredibly proud of the effort you all put in and really enjoyed teaching you.

Congrats again! Best of luck on the rest of your finals :)

(There is nothing to submit for this question)

**No more questions.**