CS 61A Structure and Interpretation of Computer Programs
Summer 2021 Final

INSTRUCTIONS

This is your exam. Complete it either at 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.

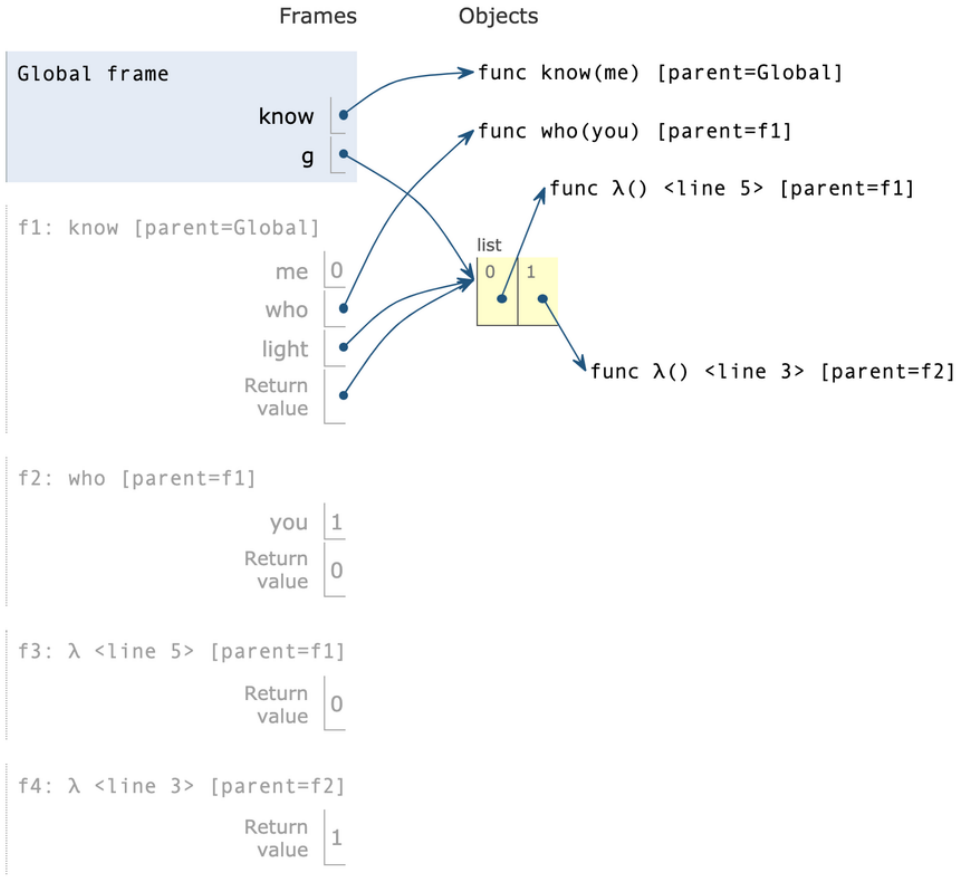
(a) What is your full name?

(b) What is your student ID number?

(c) By writing my name below, I pledge on my honor that I will abide by the rules of this exam and will neither give nor receive assistance. I understand that doing otherwise would be a disservice to my classmates, dishonor me, and could result in me failing the class.

1. (8.0 points) Next Big Thing

(a) The following environment diagram was generated by a program:



In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram.

```
def know(me):
    def who(you):
        light.__________ # (a)
        return light[-3]()
    light = [lambda: me, __________] # (b)
    return light
g = __________ # (c)
g[1](1)
g.pop(__________) # (d)
```

i. (2.0 pt) Which of these could fill in blank (a)? **Select all that apply!**

- ☐ `append([lambda: you])`
- ☐ `append(lambda you: you)`
- ☐ `append([lambda you: you])`
- ☐ `extend([lambda: you])`
- ☐ `extend([lambda you: you])`
- ☐ `append(lambda: 0)`
- ☐ `extend([lambda: 0])`
- ☐ `append(lambda: 1)`
- ☐ `extend([lambda: 1])`

ii. (2.0 pt) What line of code could go in blank (b)?

iii. (2.0 pt) Which of these could fill in blank (c)?

- ◯ `know(0)`
- ◯ `know(1)`
- ◯ `know`
- ◯ `[lambda: 0, lambda: 1]`
- ◯ `[lambda: 1, lambda: 0]`
- ◯ `[]`

iv. (2.0 pt) Which of these could fill in blank (d)? **Select all that apply!**

- ☐ `g[0]()`
- ☐ `g[1]()`
- ☐ `g[2]()`
- ☐ `g[3]()`
- ☐ `g[-1]()`
- ☐ `g[-2]()`
- ☐ `g[-3]()`
- ☐ `1`
- ☐ `0`
- ☐ `2`

2. (15.0 points) Linked Trees

(a) (7.0 points) Pruning

Definition: We call a tree *shrinking* if all of each node's branches have fewer branches than the node itself.

Write a function `prune_shrinking_tree` which takes in a tree `t`. It should return a **new tree** which is *shrinking*, by pruning off the excess *rightmost* branches of each node. **Do not prune more than the required number.**

```
def prune_shrinking_tree(t):
    """
    >>> t1 = Tree(1, [Tree(2, [Tree(4), Tree(5), Tree(6)]), Tree(3)])
    >>> prune_shrinking_tree(t1)
    Tree(1, [Tree(2, [Tree(4)]), Tree(3)])
    >>> t1
    Tree(1, [Tree(2, [Tree(4), Tree(5), Tree(6)]), Tree(3)])
    >>> t2 = Tree(1, [Tree(2), Tree(3)])
    >>> prune_shrinking_tree(t2)
    Tree(1, [Tree(2), Tree(3)])
    >>> t3 = Tree(1, [Tree(2, [Tree(6, [Tree(8)]), Tree(7)]), Tree(3), Tree(4), Tree(5)])
    >>> prune_shrinking_tree(t3)
    Tree(1, [Tree(2, [Tree(6, [Tree(8)]), Tree(7)]), Tree(3), Tree(4), Tree(5)])
    """
    def helper(t, k):
        if ___________:
             # (a)
            return ___________
                     # (b)
        else:
            prune_ct = ___________(___________, ___________)
                         # (c)         (d)          (e)
            new_branches = [helper(b, prune_ct) for b in ___________]
                                                          # (f)
            return ___________
                    # (g)
    return helper(t, len(t.branches) + 1)
```

i. Which of these could fill in blank (a)?

- ◯ `t.is_leaf()`
- ◯ `t is Link.empty`
- ◯ `is_leaf(t)`
- ◯ `t.is_leaf`
- ◯ `len(t) == 0`
- ◯ `len(t) == 1`
- ◯ `t < 10`
- ◯ `t == ''`

ii. What line of code could go in blank (b)?

iii. Which of these could fill in blank (c)?

- ◯ `min`
- ◯ `max`
- ◯ `filter`
- ◯ `map`
- ◯ `sum`

iv. Which of these could fill in blank (d)?

- ◯ `k`
- ◯ `k - 1`
- ◯ `k + 1`
- ◯ `t.label`
- ◯ `t.label - 1`
- ◯ `t.label + 1`

v. What line of code could go in blank (e)? **You may not use `min`, `max`, `filter`, `map`, or `sum` in this line.**

vi. Which of these could fill in blank (f)?

- ◯ `t.branches[k+1:]`
- ◯ `t.branches[k-1:]`
- ◯ `t.branches`
- ◯ `t.branches[:k]`
- ◯ `t.branches[:]`
- ◯ `t.branches[k:]`
- ◯ `t.branches[:k+1]`
- ◯ `t.branches[:k-1]`

vii. What line of code could go in blank (g)? **You may not using list slicing in for this blank.**

(b) (2.0 points) Linked Add

Implement a function `add_to_all` which takes in a linked list `s` and a number `x` and returns a new version of `s` with `x` added to each element.

Remember, you should not be mutating s!

You may not use the `map` function.

```
def add_to_all(s, x):
    """
    >>> s = Link(3, Link(2, Link(5)))
    >>> print(add_to_all(s, 1))
    <4 3 6>
    >>> print(add_to_all(s, 0))
    <3 2 5>
    """
    if ___________:
          # (i)
        return s
    return Link(___________, ___________)
                  # (j)          (k)
```

i. What line of code could go in blank (i)?

ii. What line of code could go in blank (j)?

iii. What line of code could go in blank (k)?

(c) (6.0 points) Digging Deep

Definition: the *depth* of a node is defined as the distance from the root of the tree to that node. For example, the *depth* of the root node is 0, and the nodes at each of the branches of that root have *depth* 1.

Implement a function `square_depths` which takes in a Tree `t` and a linked list of **increasing** numbers `depths`. It should mutate `t` by squaring the label of every node at each *depth* contained in list `depths`.

You may need to use `add_to_all` in this part.

```
def square_depths(t, depths):
    """
    >>> t1 = Tree(2, [Tree(3, [Tree(4)]), Tree(5)])
    >>> square_depths(t1, Link(0, Link(1)))
    >>> t1
    Tree(4, [Tree(9, [Tree(4)]), Tree(25)])
    >>> t2 = Tree(2, [Tree(3, [Tree(4)]), Tree(5)])
    >>> square_depths(t2, Link(2))
    >>> t2
    Tree(2, [Tree(3, [Tree(16)]), Tree(5)])
    >>> t3 = Tree(2)
    >>> square_depths(t3, Link.empty)
    >>> t3
    Tree(2)

    """
    if ___________:
          # (l)
        return
    if depths.first == 0:
        t.label = ___________
                    # (m)
        depths = ___________
                    # (n)
    for b in ___________:
                # (o)
        ____________________________
                # (p)
```

i. What line of code could go in blank (l)?

ii. What line of code could go in blank (m)?

iii. What line of code could go in blank (n)?

iv. What line of code could go in blank (o)?

v. What line of code could go in blank (p)?

3. (16.0 points) Reconstructing Ed

(a) (4.0 points) Users

The class `User` represents those who login to Edstem and make a post. Fill in the blanks for the `User`, `Student`, and `Instructor` classes such that the `__repr__` method in the `User` class returns a string that (when evaluated) creates a new instance with the same attributes as this one. See the doctests for some examples.

Instructors are admins, which means that they can view all posts, even questions not posted by themselves.

```
class User:
    """
    >>> #### test User.__repr__, Student, Instructor ####
    >>> Instructor("Alex Kassil", "alexkassil@berkeley.edu")
    Instructor('Alex Kassil', 'alexkassil@berkeley.edu')
    >>> Student("Albert Xu", "albertxu3@berkeley.edu")
    Student('Albert Xu', 'albertxu3@berkeley.edu')
    """

    admin = False
    class_name = "User"
    def __init__(self, name, email):
        self.name = name
        self.email = email

    def __repr__(self):
        return _________________
                    # (a)

class Student(User):
    class_name = _________________
                     # (b)

class Instructor(User):
    admin = _________________
                # (c)
    class_name = _________________
                     # (d)
```

i. What line of code could go in blank (a)?

ii. What line of code could go in blank (b)?

iii. What line of code could go in blank (c)?

iv. What line of code could go in blank (d)?

(b) (6.0 points) Posts

The class Post represents a single post made to Edstem. Each post is a question by default, meaning it is only visible to the person who asked it or to an admin. You can also specify the announcement instance variable, which, if True, makes a post visible to everyone. For each post, the post ID represents the post number in chronological order starting with 1. The icon is "(!)" for announcements and "(?)" for all other posts.

Fill in the blanks for Post.view such that it prints the __str__ of the post after making necessary updates to the post object. Also, fill in the blanks for Post.__str__ such that it returns a string that has the post ID, followed by a colon, a space, the icon, a space, the post title, a space, the string "<>", the email of the User who made the post, a space, the "<> Total Views:", a space, and finally the total number of times the post has been viewed. See the doctests for some examples.

```
class Post:
    """
    >>> #### setup ####
    >>> alex = Instructor("Alex Kassil", "alexkassil@berkeley.edu")
    >>> catherine = Student("Catherine Cang", "catherinecang@berkeley.edu")
    >>> Post.id = 1 # reset Post IDs
    >>> p1 = Post("Introductions", alex, True)
    >>> #### test Post.__str__ and Post.view ####
    >>> p1.views
    {}
    >>> p1.view(alex)
    1: (!) Introductions <> alexkassil@berkeley.edu <> Total Views: 1
    >>> p1.view(catherine)
    1: (!) Introductions <> alexkassil@berkeley.edu <> Total Views: 2
    >>> p1.views
    {'alexkassil@berkeley.edu': 1, 'catherinecang@berkeley.edu': 1}
    """

    id = 1
    def __init__(self, title, user, announcement=False):
        self.title = title
        self.user = user
        self.announcement = announcement
        self.views = {}
        self.total_views = 0
        self.id = _________________
                      # (e)
        _________________
             # (f)

    def __str__(self):
        if self.announcement:
            icon = "(!)"
        else:
            icon = "(?)"
        return _________________________________
                           # (g)

    def view(self, user):
        if user.email not in self.views:
            self.views[user.email] = 0
        self.views[user.email] += 1
        self.total_views = _________________
```

```
                            # (h)
print(_________________)
           # (i)
```

i. What line of code could go in blank (e)?

ii. What line of code could go in blank (f)?

iii. What line of code could go in blank (g)?

iv. What line of code could go in blank (h)?

v. What line of code could go in blank (i)?

(c) (6.0 points) Edstem

The class Edstem represents a course Q&A forum, with some set of users and posts.

Fill in add_user and add_post such that Users and Posts are saved to the Edstem instance. Then, fill in login_and_view_all. Only added users can log in. After a successful login, a user may view all posts that are visible to them (all announcements and their own questions if they are not an admin, or everything if they are an admin). Also, fill in show_stats in order to print out the __str__ of each Post, followed by a space, the string "<> Unique Viewers:", a space, and the number of unique viewers.

```
class Edstem:
    """
    >>> #### setup ####
    >>> alex = Instructor("Alex Kassil", "alexkassil@berkeley.edu")
    >>> albert = Student("Albert Xu", "albertxu3@berkeley.edu")
    >>> catherine = Student("Catherine Cang", "catherinecang@berkeley.edu")
    >>> Post.id = 1 # reset Post IDs
    >>> p1 = Post("Introductions", alex, True)
    >>> p2 = Post("HW 1 Deadline", catherine)
    >>> p3 = Post("Diagnostic Alternate", albert)
    >>> cs61a = Edstem("CS 61A")
    >>> cs61a.add_user(alex); cs61a.add_user(catherine); cs61a.add_user(albert);
    >>> cs61a.add_post(p1); cs61a.add_post(p2); cs61a.add_post(p3);
    >>> #### test Edstem.add_user ####
    >>> vanshaj = Student('Vanshaj Singhania', 'vanshaj@berkeley.edu')
    >>> len(cs61a.users)
    3
    >>> cs61a.add_user(vanshaj)
    >>> len(cs61a.users)
    4
    >>> cs61a.users['vanshaj@berkeley.edu']
    Student('Vanshaj Singhania', 'vanshaj@berkeley.edu')
    >>> #### test Edstem.login_and_view_all ####
    >>> cs61a.login_and_view_all("albertxu3@berkeley.edu")
    1: (!) Introductions <> alexkassil@berkeley.edu <> Total Views: 1
    3: (?) Diagnostic Alternate <> albertxu3@berkeley.edu <> Total Views: 1
    >>> cs61a.login_and_view_all("alexkassil@berkeley.edu")
    1: (!) Introductions <> alexkassil@berkeley.edu <> Total Views: 2
    2: (?) HW 1 Deadline <> catherinecang@berkeley.edu <> Total Views: 1
    3: (?) Diagnostic Alternate <> albertxu3@berkeley.edu <> Total Views: 2
    >>> try:
    ...     cs61a.login_and_view_all("timothyktu@berkeley.edu")
    ... except:
    ...     print("This user is not enrolled!")
    ...
    This user is not enrolled!
    """

    def __init__(self, course):
        self.course = course
        self.users = {}
        self.posts = []

    def add_user(self, user):
        ---------------------------------
                    # (j)
```

```
def add_post(self, post):
    self.posts.append(post)

def login_and_view_all(self, email):
    assert _________________
                # (k)
    for post in self.posts:
        if _________________ or _________________ or _________________:
            #    (l)                 (m)                 (n)
            post.view(self.users[email])

def show_stats(self):
    for post in self.posts:
        print(post, "<> Unique Viewers:", len(_________________))
                                                    # (o)
```

i. What line of code could go in blank (j)?

ii. What line of code could go in blank (k)?

iii. What line of code could go in blank (l)?

iv. What line of code could go in blank (m)?

v. What line of code could go in blank (n)?

vi. What line of code could go in blank (o)?

4. (11.0 points) Ed Analysis

Ed is a Q&A Forum. The `students` table describes the Name and Email for each student enrolled in the CS 61A Ed. The `posts` table describes the email (of the poster), title, and timestamp for each post made on the CS 61A Ed. Emails are unique to students – each email can only be associated with one student.

You may assume that all timestamps are all unique, but titles in posts may not be unique.

```
CREATE TABLE students AS
SELECT "Amritansh Saraf" AS name, "amritansh@cs61a.org" AS email UNION
SELECT "Cindy Lin"              , "cclin@cs61a.org"             UNION
SELECT "Vanshaj Singhania"      , "vanshaj@cs61a.org"           UNION
SELECT "Marie Chorpita"         , "chorpita@cs61a.org";
```

```
CREATE TABLE posts AS
SELECT "vanshaj@cs61a.org" AS email, "Scheme Project EC" AS title, 1627774233 as timestamp UNION
SELECT "chorpita@cs61a.org"        , "Signing up for Tutoring"   , 1627775133 UNION
SELECT "cclin@cs61a.org"           , "Doctors Hate Her!"         , 1627774133 UNION
SELECT "cclin@cs61a.org"           , "Doctors Hate Her!"         , 1627774135 UNION
SELECT "vanshaj@cs61a.org"         , "Scheme Project EC"         , 1627784233;
```

(a) (2.0 points) Broken Timestamps

Definition: A *timestamp* represents a moment in time based on how many seconds (not including leap seconds) have passed since 00:00:00 UTC on January 1, 1970. So, for example, 00:01:00 UTC on January 1, 1970 would be represented as the *timestamp* `60`. The start of the regular exam time for the Summer 2021 61A final exam, 17:00:00 UTC-7 on August 12, 2021, would be represented as `1628812800`.

The timestamps in the `posts` table are all a week too early, and we need to fix them! Write a SQL query that selects the email, title, and timestamp columns, while adding 7 days to the timestamp.

Hint: there are 604800 seconds in 7 days.

You may use both commas and `AND` inside your answers.

Output for the sample table:

cclin@cs61a.org	Doctors Hate Her!	1628378933
cclin@cs61a.org	Doctors Hate Her!	1628378935
chorpita@cs61a.org	Signing up for Tutoring	1628379933
vanshaj@cs61a.org	Scheme Project EC	1628379033
vanshaj@cs61a.org	Scheme Project EC	1628389033

```
SELECT ___________;
--          (a)
```

i. What line of code could fill in the blank (a)?

(b) (4.0 points) First

Complete a SQL query that selects a three-column table with the student `name`, `title`, and `timestamp` of the *first post* made by each student.

You may use both commas and `AND` inside your answers.

Output for the sample table:

Cindy Lin	Doctors Hate Her!	1627774133
Marie Chorpita	Signing up for Tutoring	1627775133
Vanshaj Singhania	Scheme Project EC	1627774233

```
SELECT ___________ FROM students AS a, posts AS p WHERE ___________ GROUP BY ___________;
--          (b)                                          (c)                  (d)
```

i. What line of code could fill in the blank (b)?

ii. What line of code could fill in the blank (c)?

iii. What line of code could fill in the blank (d)?

(c) (5.0 points) Duplicate Detection

Definition: *Duplicate Posts* are two posts that are made by the same student with the same `title` and within 10 seconds of each other. Recall that the *timestamp* values are measured in seconds.

Create a four-column table of author's `name`, post `title`, and both `timestamp`s for each pair of duplicate posts in the `posts` table. Only select pairs where the first post occurs *less than or equal to* 10 seconds before the second post (e.g., don't include duplicates in the other direction)

You may not use `AND`, `NOT`, `OR` or `AS` in any of your below answers.

Output for the sample table:

Cindy Lin	Doctors Hate Her!	1627774133	1627774135

```
SELECT z.name, x.title, x.timestamp, y.timestamp
    FROM __________ AS x, __________ AS y, __________ AS z
--           (e)               (f)               (g)
    WHERE __________ AND __________ AND __________ AND __________ AND __________;
--           (h)               (i)               (j)               (k)               (l)
```

i. What line of code could fill in the blank (e)?

ii. What line of code could fill in the blank (f)?

iii. What line of code could fill in the blank (g)?

iv. What line of code could fill in the blank (h)?

v. What line of code could fill in the blank (i)?

vi. What line of code could fill in the blank (j)?

vii. What line of code could fill in the blank (k)?

viii. What line of code could fill in the blank (l)?

5. (13.0 points) Scheme

(a) (4.0 points) If Fibonacci is so great...

Definition: Each element of the `fibonacci2` sequence is defined as twice the absolute value of the difference between the previous two elements. Assume that the 0th element of the `fibonacci2` sequence is `0`, and the 1st element is `1`.

Implement the function `fib2`, which takes in one parameter `n`, a non-negative integer, and returns the `n`th element of the `fibonacci2` sequence.

Reminder: Scheme has a built in procedure `abs` which returns the absolute value of the argument that is passed in.

```
(define (fib2 n)
    (if ___________ n
          ; (a)
          (___________ ___________ (___________ (- ___________ ___________)))))
             ; (b)         (c)          (d)            (e)         (f)
(expect (fib2 0) 0)
(expect (fib2 1) 1)
(expect (fib2 2) 2)
(expect (fib2 3) 2)
(expect (fib2 4) 0)
(expect (fib2 5) 4)
```

i. What line of code could fill in the blank (a)?

ii. Which of these could fill in blank (b)?

- ◯ `*`
- ◯ `-`
- ◯ `square`
- ◯ `fib2`
- ◯ `fib`

iii. What line of code could fill in the blank (c)?

iv. What line of code could fill in the blank (d)?

v. What line of code could fill in the blank (e)?

vi. What line of code could fill in the blank (f)?

(b) (7.0 points) The (n-1)-al Countdown

Definition: The *countdown sequence* of a number `n` is the sequence starting at `n` and descending to 0. For example, the *countdown sequence* of 3 is `3 2 1 0`.

Implement a function `countdowns` which takes in a scheme list `lst` of non-negative integers and returns a list which is the concatenation of the *countdown sequences* of each element in `lst`.

```
(define (countdowns lst)
   (cond ((null? lst) ___________)
                         ; (k)
         ((> ___________ 0) (cons (car lst)
               ; (l))
             (countdowns ___________)))
                          ; (m)
         (else (cons 0 ___________))))
                          ; (o)
(expect (countdowns '(3)) (3 2 1 0))
(expect (countdowns '(2 0 3)) (2 1 0 0 3 2 1 0))
(expect (countdowns '()) ())
```

i. What line of code could fill in the blank (k)?

ii. What line of code could fill in the blank (l)?

iii. What line of code could fill in the blank (m)?

iv. What line of code could fill in the blank (n)?

(c) (2.0 points) Tail Recursion?

i. We would like to modify the implementation of `countdowns` to make the function tail recursive. This could include modifications to the skeleton code. Is this possible?

◯ `countdowns` is not tail recursive but can be made tail recursive

◯ `countdowns` cannot be made tail recursive

◯ `countdowns` is already tail recursive

6. (8.0 points) All Links

(a) Implement a generator function `all_links`, which takes in `nums`, a list of equal-length lists. `all_links` should yield all linked lists `s` that can be constructed such that the first element of `s` is the first element of one of the lists in `nums`, the second element of `s` is the second element of one of the lists in `nums`, and so on. Lists can be yielded in any order. You can assume that `nums` is a non-empty list.

For example, for the second doctest `all_links([[0, 2], [1, 3]])`, there are four total linked lists we should yield. `Link(0, Link(2))` is generated from using the first element from the first list and the second element from the first list. `Link(0, Link(3))` is generated with the first element from the first list and the second element from the second list. `Link(1, Link(2))` and `Link(1, Link(3))` get the first element from the second list and the second element from the first and second lists respectively.

```
def all_links(nums):
    """
    >>> list(all_links([[0], [1], [2]]))
    [Link(0), Link(1), Link(2)]
    >>> list(all_links([[0, 2], [1, 3]]))
    [Link(0, Link(2)), Link(0, Link(3)), Link(1, Link(2)), Link(1, Link(3))]
    """
    if len(nums[0]) == 0:
        ___________
          # (a)
    else:
        rests = [___________ for x in nums]
                    # (b)
        for first in [x[0] for x in nums]:
            for item in ___________:
                        # (c)
                ___________
                  # (d)
```

i. What line of code could fill in the blank (a)?

ii. Which of these could fill in blank (b)?

- ◯ `x[1:]`
- ◯ `x[1]`
- ◯ `nums[1]`
- ◯ `nums[1:]`
- ◯ `nums[-1]`
- ◯ `nums[:-1]`
- ◯ `x[-1]`
- ◯ `x[:-1]`

iii. What line of code could fill in the blank (c)?

iv. What line of code could fill in the blank (d)?

7. (9.0 points) Merger

(a) (9.0 points)

Consider the function `merger`, which returns the result of merging all the terms in a sequence using `f`, a two-argument function. The function `f` is applied to the first and second elements until there is only one element in the subsequence.
For example, merging the sequence `1, 2, 3` with the function `lambda x, y: x + y` would return the result `6`.

Now consider the sequence of ascending integers: `1, 2, 3, ... k` and all of its unique ascending subsequences of at least length two. For example, the sequence `[1,2,3,4]` has the following unique ascending subsequences of length at least two: `[1,2], [1,3], [1,4], [1,2,3], [1,2,4], [1,3,4], [2,3], [2,4], [2,3,4], [3,4], [1,2,3,4]`. Our goal is to count the number of subsequences of `merger` that result in exactly `N` when merging with `f`.

For the first doctest, `f` is `lambda x, y: x + y`, `N` is 6, and `K` is 4, so only `[1,2,3]` and `[2,4]` add up to 6, meaning `count_merger(6, 4, lambda x, y: x + y)` returns 2.

Write a function `count_merger` that returns the number of ways to make exactly `N` using at least two unique, ascending integers from range `1` to `K` (inclusive) and a function `f` which is a two-argument function.

```
def count_merger(n,k,f):
    """
    Returns the number of ways to make exactly N using at least two unique,
    ascending integers from range 1 to K (inclusive) and a function f that
    accepts two integers as arguments and returns an integer.
    Assume K >= 1.
    >>> from operator import add, mul, sub
    >>> add1_mult = lambda x, y: (x + 1) * y
    >>> count_merger(6, 4, lambda x, y: x + y) # 1 + 2 + 3 = 6, 2 + 4 = 6
    2
    >>> count_merger(36, 6, mul) # 1 * 2 * 3 * 6; 2 * 3 * 6
    2
    >>> count_merger(36, 6, add)
    0
    >>> count_merger(36, 6, add1_mult) # (5 + 1) * 6
    1
    >>> count_merger(24, 3, lambda x, y: 24) # f(1, 2); f(f(1, 2), 3); f(1, 3); f(2, 3)
    4
    >>> count_merger(1, 1, lambda x, y: x)
    0
    >>> count_merger(-2, 5, sub) # 1 - 3; 2 - 4; 3 - 5
    3
    """
    def helper(start, i):
        if i > k:
            return 0
        a = helper(start, i + 1)
        b = helper(______, i + 1)
                    # (a)
        if ___________ == n:
             # (b)
            return ___________
                     # (c)
        else:
            return ___________
                     # (d)
```

```
total = 0
i = 1
while ____________:
          # (e)
    total += ____________
               # (f)
    i += 1
return total
```

i. What line of code could go in blank (a)?

ii. What line of code could go in blank (b)?

- ◯ `start`
- ◯ `start + 1`
- ◯ `helper(start, i)`
- ◯ `helper(start, i + 1)`
- ◯ `f(start, i)`
- ◯ `f(start, i + 1)`

iii. What line of code could go in blank (c)?

iv. What line of code could go in blank (d)?

v. What line of code could go in blank (e)?

- ◯ `i > 0`
- ◯ `k > 0`
- ◯ `i >= 0`
- ◯ `k >= 0`
- ◯ `i > k`
- ◯ `i < k`
- ◯ `i >= k`
- ◯ `i <= k`

vi. Which of these could fill in blank (f)?

- ◯ `helper(i, i)`
- ◯ `helper(i, i + 1)`
- ◯ `helper(i, total + 1)`
- ◯ `helper(total, i)`
- ◯ `helper(total, i + 1)`
- ◯ `helper(total, total + 1)`

8. Extra Sus

(a) Amogus's Dilemma

In this extra credit problem, you may choose one of two options.

- Mark the choice of "Impostor" and write a positive integer in the blank below. The one student who writes the smallest unique positive integer will receive three (3) extra credit points but only if fewer than 90% of students choose the next option.
- Mark the choice of "Crewmate". If at least 90% of students choose this option, all students who chose this option will receive one (1) extra credit point and those who marked the choice to "Impostor" will receive zero (0) extra credit points.

i.

◯ Impostor

◯ Crewmate

ii. If you marked Impostor, type in a number.

No more questions.