DS C88C Fall 2023 Computational Structures in Data Science

FINAL

INSTRUCTIONS

- **Do NOT open the exam until you are instructed to do so!**
- You **must not** collaborate with anyone inside or outside of C88C.
- You **must not** use any internet resources to answer the questions.
- If you are taking an online exam, 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.
- When a question specifies that you must rewrite the completed function, you should **not** recopy the doctests.
- The exam is closed book, closed computer, closed calculator, except your hand-written 8.5" x 11" cheat sheets of your own creation and the official C88C Reference Sheet

Full Name	
Student ID Number	
Official Berkeley Email (@berkeley.edu)	
What room are you in?	
Name of the person to your left	
Name of the person to your right	
By my signature, I certify that all the work on this exam is my own, and I will not discuss it with anyone until exam session is over. **(please sign)**	

POLICIES & CLARIFICATIONS

- If you need to use the restroom, bring your phone and exam to the front of the room.
- For fill-in-the-blank coding problems, we will only grade work written in the provided blanks.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.
    - **Please write your SID at the top of each page!**
    - **You must include all answers within the boxes.**
    - If you must write outside the box, please draw an arrow.
    - Use the blank space as scratch paper to work out your solutions.

1. (10.0 points) What Would Python Do (WWPD)

For each expression below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write "Error" (if any lines are displayed before the error, include those in your output). If a function is returned, write "Function". If the value "None" is returned, write "None".

NOTE: Assume each part is executed *in order*. Previous lines DO impact the current expression. (i.e., part B assumes part A was executed, as so on.)

```
f = lambda x, y: 10 - (x + y)
def tricky(f, n):
    if n < 10:
        n *= 2
    y = 15
    return lambda x, y: f(x, n) + y
```

(a) (2.0 pt)

```
>>> tricky(f, 5)
```

(b) (2.0 pt)

```
>>> tricky(f, 5)(3)
```

(c) (2.0 pt)

```
>>> tricky(f, 5)(3, 20)
```

(d) (1.0 pt)

```
def outer_function():
    a = 5
    def inner_function():
        a = 10
        return a
    b = inner_function()
    print(f"Inner: {b}, Outer: {a}")

outer_function()
```

◯ `Inner: 5, Outer: 10`

◯ `Inner: 10, Outer: 5`

◯ `Inner: 10, Outer: 10`

◯ `The code will result in an error.`

(e) (1.0 pt) Given a list of numbers, which code snippet uses `filter` to return a list with all negative numbers removed? (Note: `filter` in Python returns an iterator that needs to be converted to a list)

```
numbers = [4, -1, -3, 2, 0, -5, 8]
```

◯ `filter(lambda x: x < 0, numbers)`

◯ `filter(lambda x: x > 0, numbers)`

◯ `list(filter(lambda x: x < 0, numbers))`

◯ `list(filter(lambda x: x >= 0, numbers))`

(f) (1.0 pt)

```
def square(number):
    try:
        return number ** 2
    except TypeError:
        return "hello"

print(square("two"))
```

◯ It will print "two"

◯ It will print 4

◯ It will print "hello"

◯ It will display a TypeError

(g) (1.0 pt)

```
def check_even(number):
    return "Even" if number % 2 == 0 else "Odd"

numbers = [1, 2, 3, 4]
result = map(check_even, numbers)
print(list(result))
```

◯ [False, True, False, True]

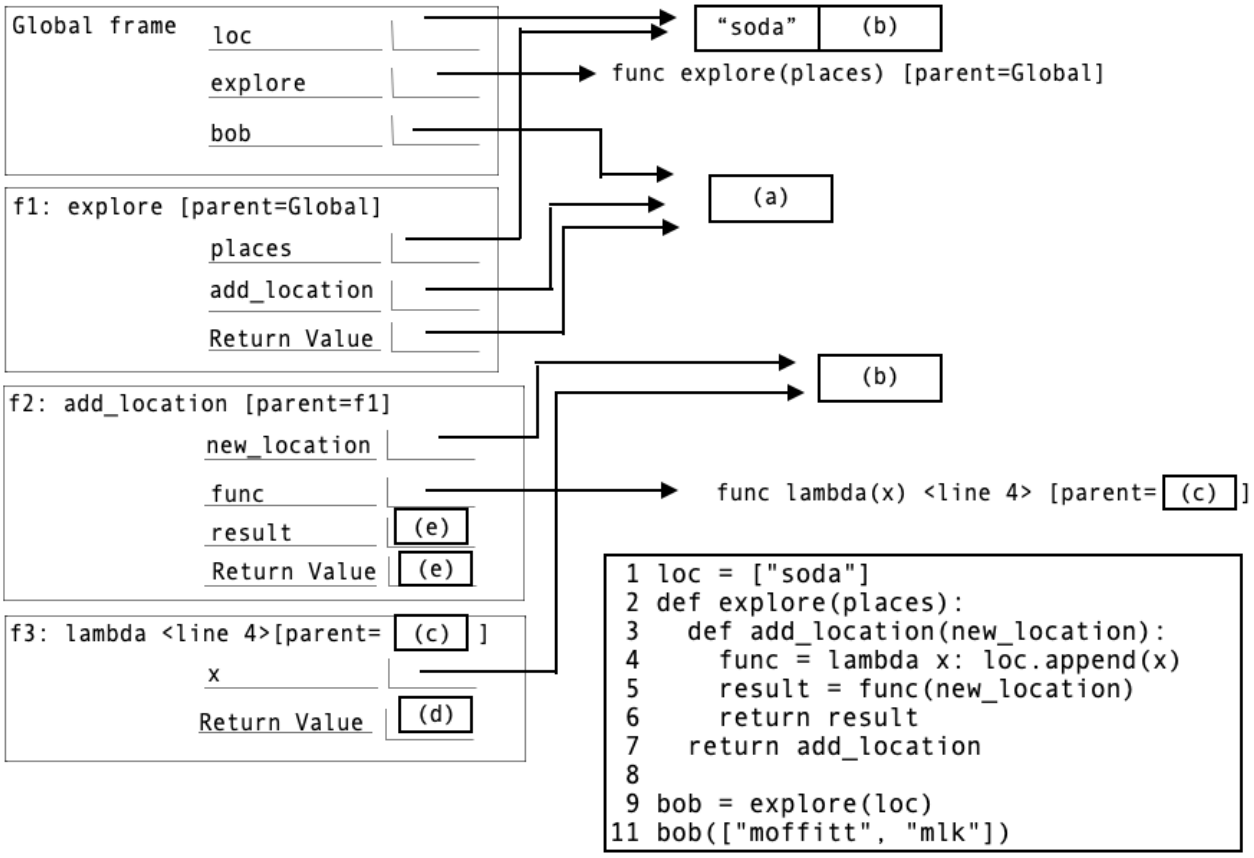
◯ [Odd, Even, Odd, Even]

◯ [1, 2, 3, 4]

◯ [None, None, None, None]

2. (10.0 points) Let's Explore!

Fill in the blanks to complete the environment diagram. All the code used is in the box to the right, and the code runs to completion. Boxes with the same label will have the same value.



`explore` **environment diagram**

(a) (2.0 pt) What is value of box **(a)**?

- ◯ `func add_location(new_location) [parent=f1]`
- ◯ `func add_location(new_location) [parent=Global]`
- ◯ `func explore(places) [parent=f1]`
- ◯ `func explore(places) [parent=Global]`
- ◯ `["moffitt", "mlk"]`
- ◯ `["soda", "moffitt", "mlk"]`

(b) (2.0 pt) What is the second element in the list `loc`, item **(b)**?

(c) (2.0 pt) What is the parent of `lambda` function in frame 3, item **(c)**?

- ◯ `Global`
- ◯ `f1`
- ◯ `f2`
- ◯ `f3`

(d) (2.0 pt) What is the return value of the lambda function in f3, item **(d)**?

- ◯ An arrow pointing to the list `loc`
- ◯ `["soda", ["moffitt", "mlk"] ]`
- ◯ `["soda", "moffitt", "mlk"]`
- ◯ `["soda"]`
- ◯ `None`

(e) (2.0 pt) What is value of `result` and the Return Value of `add_location` in f2, item **(e)**?

- ◯ `None`
- ◯ `Error`
- ◯ An arrow pointing to the box **(a)**
- ◯ An arrow pointing to the list `loc`
- ◯ `"soda"`

3. (4.0 points) A Broken Phone...Book

(a) (2.0 pt) Siya lent her phone book to Sanjana. Upon receiving it back, she notices that Sanjana has tampered with some of the contacts, either adding or deleting digits from their phone numbers.

The phone book after Sanjana has tampered with it looks like this:

```
phonebook = [ {"Oski": 4084170388}, {"Carol": 609627}, {"Michael": 510921913888} ]
```

Siya writes a program that will output a list of all her compromised contacts. A contact is valid if the phone number is exactly 10 digits long, and is compromised otherwise.

She writes the program code shown below and expects it to output: `ans = ["Carol", "Michael"]`

Before she runs her code, she asks you to help her debug.

```
def funky_phonebook(phonebook):
  ans = []
  for item in phonebook:
    for k in item:
      if len(str(item[0])) < 10 or len(str(item[0])) > 10:
        ans.append(k)
  return ans
```

Select the option which describes the result of this code.

◯ The code errors and will return a key error

◯ The code errors and will return a syntax error

◯ The code is incorrect and will return a list of valid contacts

◯ The code is correct and will return a list of compromised contacts

(b) (2.0 pt) Her mom marvels at her newfound coding abilities and proposes a challenge. She wants Siya to write a program that will take in a phone book with valid phone numbers and give all the contacts a nickname. For the purposes of this question, nicknames are the first three letters of a person's actual name. Assume that all names are at least 3 letters long.

Note that the phone book is formatted differently from part a. The desired output is as follows:

```
new_phonebook = {"Varun": 732789744, "Derrick": 510994933, "Oski": 408984443}
>>> nicknames(new_phonebook)
>>> {"Var": 732789744, "Der": 510994933, "Osk": 408984443}
```

Once again, Siya writes code that she needs your feedback on:

```
def nicknames(original_dict):
  names = {}
  for d in original_dict.keys():
    x = original_dict.pop(d)
    names[ d[:2] ] = x
  return names
```

How will this code behave?

◯ The code block runs as expected, no changes needed.

◯ The code block errors and does not run.

◯ The code block runs but does not run as expected.

4. (6.0 points) Your Mileage May Vary

You're job is to analyze environmental data about some cars. You're given a Python list `mpgs` that contains the *mpg* (miles per gallon) of many different cars, and the goal is to calculate the *variance* of the data set. As this problem contains many sub-parts, you may assume that any function or variable that is given for you to define is correct when referenced later on in the problem.

Initially, we have the list `mpgs` below:

```
>>> mpgs = [19, 22, 25, 16, 33, 35 ....]
```

(a) (1.0 pt) Write a function `avg` that takes a list of numbers as an input and outputs the mean of these numbers. The mean is defined as the sum of all of the elements of the list divided by the number of elements in the list. You may use any valid method that fits in the space provided.

```
>>> mpgs = [1, 2, 3, 4, 5]
>>> mean = avg(mpgs)
>>> mean
3
```

```
def avg(lst):

    return _____________________________________________
```

(b) (2.0 pt) Filter the `mpgs` list, using `filter` to keep all values greater than or equal to 20.

```
>>> mpgs = [11, 26, 36, 14, 5]
>>> mpgs = list(filter(______, _____))
>>> mpgs
[26, 36]
```

```
mpgs = list(filter(_____________________________________________,

                   _____________________________________________))
```

(c) (2.0 pt) Using `map`, subtract the average of the data set (`mean`) from each element in the data set and square this difference. Assign the result to a variable `squared_difference`.

```
>>> mpgs = [1, 2, 3, 4, 5]
>>> mean = avg(mpgs)
>>> squared_difference = list(map(_____, _____))
>>> squared_difference[0] # (1 - 3) = 2, 2 * 2 = 4
4
>>> squared_difference
[4, 1, 0, 1, 4]
```

```
squared_difference = list(map(_____________________________________________,

                          _____________________________________________))
```

(d) (1.0 pt) Finally, find the mean (average) of the `squared_difference` list, assigning it to `variance`.

```
variance = _____________________________________________
```

5. (10.0 points) CineDict

(a) (2.0 pt) Complete `create_movie` which creates a new movie dictionary from title, director, and release year.

```
>>> create_movie("Inception", "Christopher Nolan", 2010)
    {'title': 'Inception', 'director': 'Christopher Nolan', 'release_year': 2010}
```

```
def create_movie(title, director, year):

    return _____
```

(b) (5.0 pt) Update the movie catalog with the new movie dictionary. If the director does not exist in the catalog, add them.

```
def add_movie_to_catalog(movie_catalog, movie):
    """
    >>> movie_catalog = {}
    >>> inception = create_movie("Inception", "Christopher Nolan", 2010)
    >>> add_movie_to_catalog(movie_catalog, inception)
    {'Christopher Nolan': [{'title': 'Inception',
            'director': 'Christopher Nolan',
            'release_year': 2010}]}
    """
```

```
def add_movie_to_catalog(movie_catalog, movie):

    director = movie[_____]

    if _____ not in _____:

        movie_catalog[_____] = [movie]
    else:

        movie_catalog[director].append(_____)
    return movie_catalog
```

(c) (3.0 pt) Retrieve a list of all movies by a given director.

Hint: `dict.get(key, value)` returns `dict[key]` if `key` exists, otherwise returns `value`.

```
def get_movies_by_director(movie_catalog, director):
    """
    Retrieves a list of all movie dictionaries by a given director from the movie_catalog.
    >>> movie_catalog = {'Christopher Nolan': [{'title': 'Inception',
                                               'director': 'Christopher Nolan',
                                               'release_year': 2010 }]}
    >>> get_movies_by_director(movie_catalog, "Christopher Nolan")
    [{'title': 'Inception', 'director': 'Christopher Nolan', 'release_year': 2010}]
    >>> get_movies_by_director(movie_catalog, "Quentin Tarantino")
    []
    """
```

```
def get_movies_by_director(movie_catalog, director):

    return _____.get(

        _____, _____)
```

6. (10.0 points) Composing Trees with Trees

In this question, you will manipulate a Tree which contains a function, in addtion to the value for each node.

A *compound function*, $h(x)$, is the composition of two functions, $f(x)$ and $g(x)$. Suppose we have two functions: **f(x) = x + 2** and **g(x) = 3 * x**.

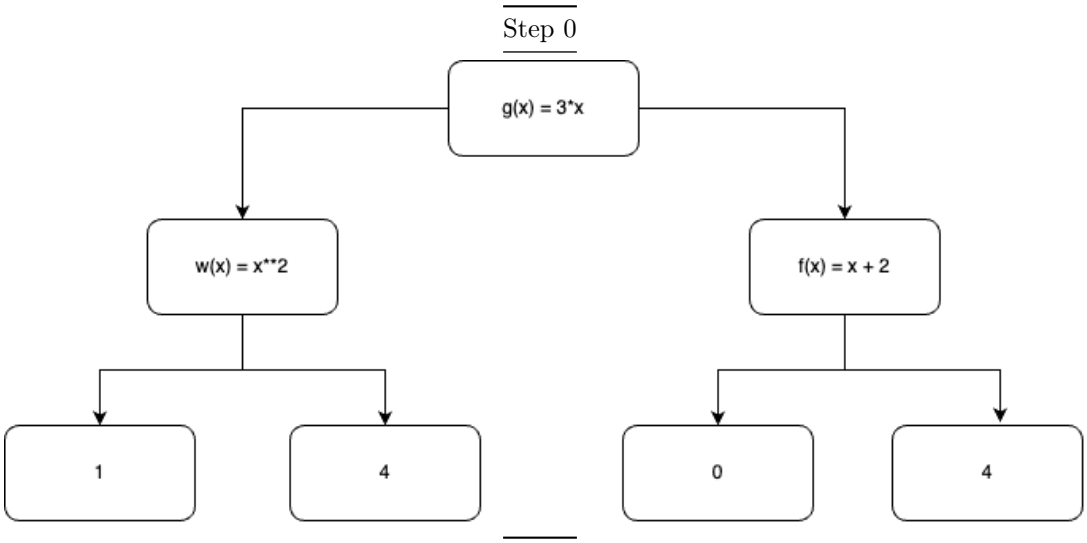
Thus the compound function `h` is **h(x) = f(g(x))** is $(3 * x) + 2$

We have added an instance attribute to the `Tree` class, `fn`, which is a function. This function takes in only one integer as an argument and returns an integer. Here is the implementation of the `Tree` class with this new `fn` attribute. You may assume that no other changes were made from the Tree class you've used in CS88.

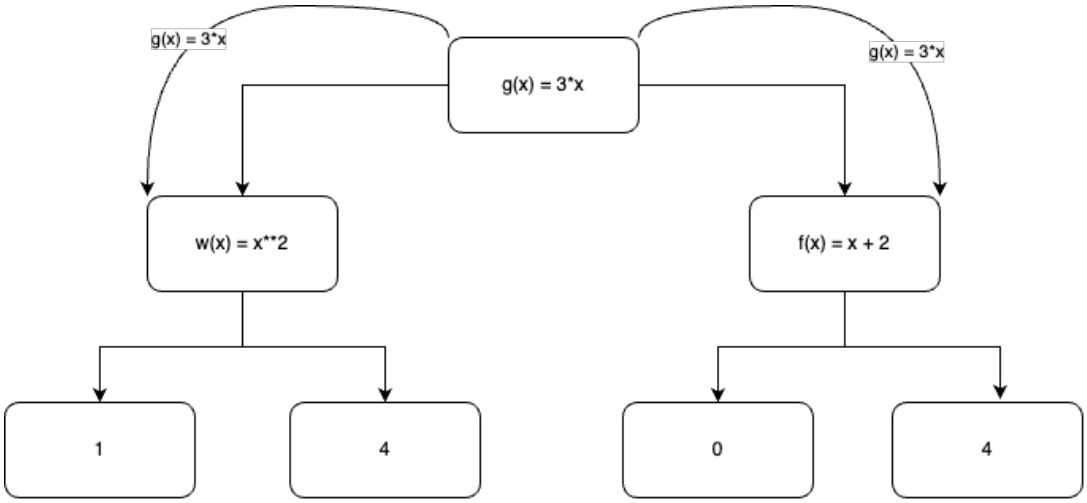
```
class Tree:
    def __init__(self, value, fn, branches=()):
        self.value = value
        self.fn = fn # NEW!
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

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

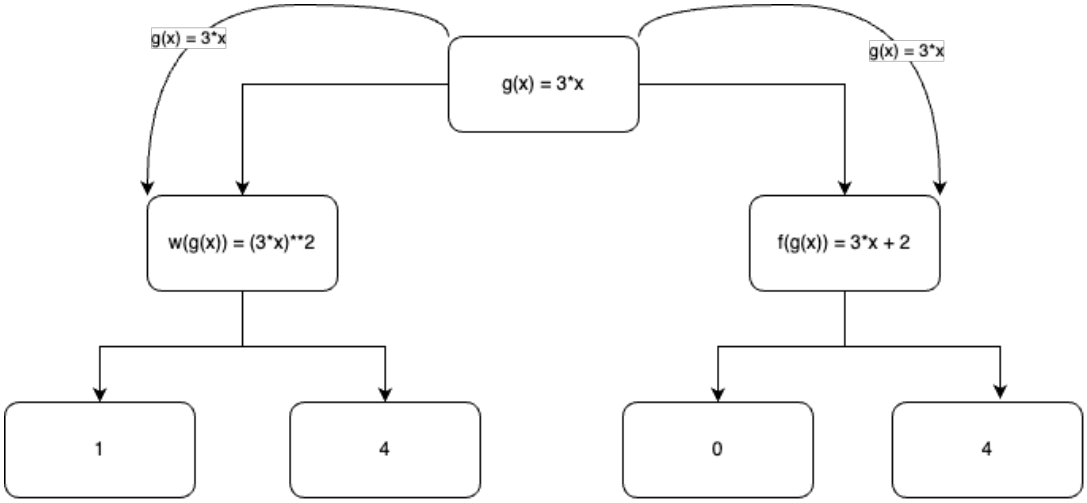
Complete the function, `compound`, that composes functions each function at every intermediate node, with their parent node's function, and applies the compound function to the value field of the tree's leaf node, modifying each leaf node in place. It is important to note that every leaf node's function will be `lambda x: x` i.e., the output is equal to the input. You may modify the `tree.fn` and `tree.value` attributes for any node in the argument tree that is passed into `compound`. Only the value field of the leaf node will be checked for accuracy. (In a Tree, remember that each node has only one parent node.)



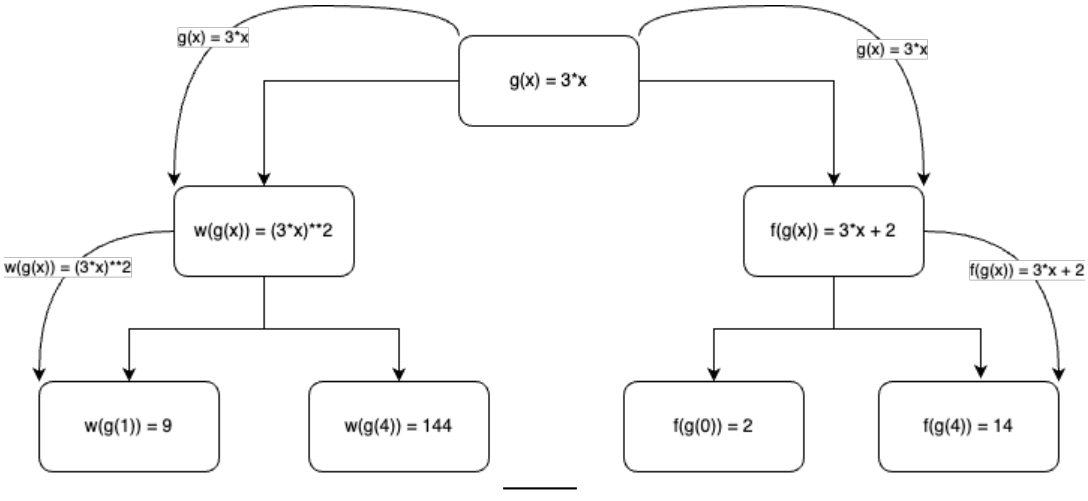
Step 1



Step 2



Step 3



For example, if one tree traversal (from the root node to the leaf node) is `(lambda x: 3 * x)` -> `(lambda x: x + 2)` -> `4`. The compound function is `(lambda x: 3*x + 2)`. This compound function is then applied to the value field of the leaf node: `4`, giving us a final value of `14`.

Referencing the code and examples above, implement the `compound` function below.

Hint: The child's function is the outer function (`f`), and the parent function is the inner function (`g`) of the composition.

```
def compound(tree):
    if _____(a)_____:
        tree.value = ____(b)____
    else:
        g = tree.fn
        for branch in ____(c)____:
            f = ____(d)____
            h = lambda x: ____(e)____
            branch.fn = h
            compound(branch)
```

(a) (2.0 pt) Fill in blank (a).

(b) (2.0 pt) Fill in blank (b).

(c) (2.0 pt) Fill in blank (c).

(d) (2.0 pt) Fill in blank (d).

(e) (2.0 pt) Fill in blank (e).

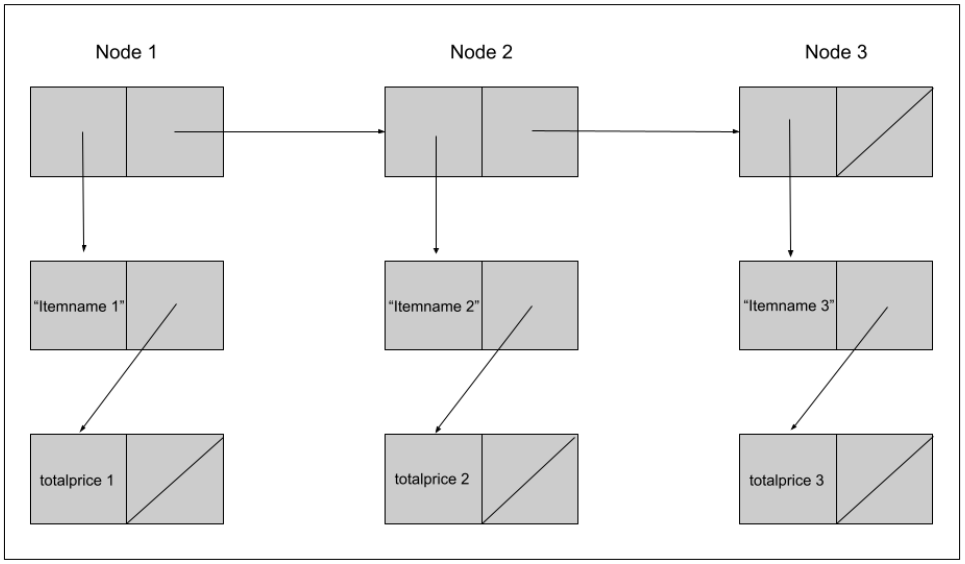
7. (8.0 points) Shopping List 2

You are going to buy groceries at the store (again). This time, you used the `aggregate_price` function you wrote for midterm 1 and have a shopping list that is a `dictionary` of the following format:

`{"itemname 1": totalprice 1, "itemname 2": totalprice 2, "itemname 3": totalprice 3}`

where `"itemname"` is a string that refers to the name of the item and `totalprice` is an int that refers to the total price of that item.

You want to store these results in a linked list. Define a function that takes in a dictionary of the aforementioned format and returns a linked list. The value of each node should be a 2-node linked list where the first node stores the item's name and the second node stores the aggregated price for the item. For example:



Linked Lists Question Diagram

`node` refers to each node (link) of the main linked list.

```
def lnk_aggregate(d):
    """
    >>> shopping_dict = {"apple": 45, "banana": 28, "milk": 3, "carrots": 15}
    >>> res = link_aggregate(shopping_dict)
    >>> res
    Link( Link("apple", Link(45)),
          Link( Link("banana", Link(28)),
              Link( Link("milk", Link(3)),
                Link( Link("carrots", Link(15)) )
            )
        )
    )
    """
    agg_lnk = None
    curr_lnk = None
    for item_name, total_price in d.items():
        node_lnk = _____________________________________________
        if agg_lnk == None:
            agg_lnk = _____________________________________________
            curr_lnk = _____________________________________________
        else:
            _____________________________________________________
            _____________________________________________________
    return agg_lnk
```

(a) (8.0 pt) Complete the function defintion. You may assume that the dictionary, `d`, passed in will be of the specified format.

```
def lnk_aggregate(d):
    agg_lnk = None
    curr_lnk = None
    for item_name, total_price in d.items():

        node_lnk = ________________________________________________
        if agg_lnk == None:

            agg_lnk = ____________________________________________

            curr_lnk = ___________________________________________
        else:

            ______________________________________________________

            ______________________________________________________
    return agg_lnk
```

8. (14.0 points) Mario Kart

Mario Kart is a popular racing video game. Let's implement some classes to make the game work. First, write a `Vehicle` class. To start you off, we have provided you with the `__init__` method:

```
class Vehicle:
    def __init__(self, speed, acceleration, weight):
        self.speed = speed
        self.acceleration = acceleration
        self.weight = weight

        self.items = []

    # More methods to be implemented below
```

(a) (2.0 pt) Over the course of a race, vehicles can pick up items and throw them to slow down their opponents.

First, write a `Vehicle` method called `pick_up_item`, which has 2 arguments: `item_name` (a string) and `item_weight` (an integer). When a driver picks up an item, the `weight` of their vehicle increases by `item_weight` and the `item_name` gets added to the `Vehicle`'s list of items.

```
def pick_up_item(self, item_name, item_weight):
    """
    >>> v = Vehicle(10, 10, 15)
    >>> v.pick_up_item('red shell', 1)
    >>> v.weight
    16
    >>> v.items
    ['red shell']
    >>> v.pick_up_item('banana', 2)
    >>> v.weight
    18
    >>> v.items
    ['red shell', 'banana']
    """
    ___(a)___
    ___(b)___
```

What should go in blank (a)?

- ◯ `self.items.append(item_weight)`
- ◯ `self.items.append(self.weight)`
- ◯ `self.speed += item_weight`
- ◯ `self.acceleration += item_weight`
- ◯ `self.weight += item_weight`

(b) (2.0 pt) What should go in blank (b)?

- ◯ `self.items.append(item_name)`
- ◯ `self.items.append(item_weight)`
- ◯ `self.items += item_name`
- ◯ `self.weight += item_name`
- ◯ `self.items += [item_weight]`

(c) (2.0 pt) Next, implement the `use_item` method which takes in a list of `Vehicle` objects. The order of the list represents the positions of each vehicle in the race. For example, the vehicle at index 0 is **behind** the vehicle at index 1.

When `use_item` is called, the item at the end of the `items` list of the vehicle is removed. Additionally, the vehicle in front of the current vehicle has its speed reduced by 1.

You may assume the `Vehicle` that calls `use_item` will always be in the `vehicles` list, have at least 1 item, and have another `Vehicle` in front of it. Additionally, assume the speed of a `Vehicle` after being hit by an item will remain positive.

```
def use_item(self, vehicles):
    """
    >>> second_place = Vehicle(5, 5, 5)
    >>> first_place = Vehicle(10, 10, 10)
    >>> vehicles = [second_place, first_place]
    >>> second_place.pick_up_item('green shell', 1)
    >>> second_place.items
    ['green shell']
    >>> second_place.use_item(vehicles)
    >>> second_place.items
    []
    >>> first_place.speed
    9
    """
    self.items.pop()
    for i in range(len(vehicles)):
        if ___(c)___:
            curr_location = i
            break
    next_location = curr_location + 1
    if next_location < len(vehicles):
        next_vehicle = ___(d)___
        ___(e)___
```

What should fill in blank (c)?

(d) (2.0 pt) What should fill in blank (d)?

(e) (2.0 pt) What should fill in blank (e)?

(f) (4.0 pt) In Mario Kart 8, there are different types of vehicles. To represent this in code, we have the `Bike` class, which is identical to the `Vehicle` class except it overrides the `pick_up_item` method.

When a `Bike` picks up an item, its `acceleration` is decreased by half the weight of the item it picks up. Additionally, the `Bike`'s `weight` and `items` attributes are modified just like in the original `pick_up_item` method. Implement this below.

```
class Bike(Vehicle):
    def pick_up_item(self, item_name, item_weight):
        """
        >> b = Bike(10, 10, 10)
        >> b.pick_up_item('blue shell', 4)
        >> b.weight
        14
        >> b.items
        ['blue shell']
        >> b.acceleration
        8.0
        """

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

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

9. (10.0 points) Generate Factors

We are writing a program that will continuously generate the factors of our input. A factor of a number divides the given number evenly or exactly.

(a) (8.0 pt) Implement the generator function `generate_factors`. The input `k` is a positive integer. It continuously yields the next factor of `k` from smallest to largest. If all of the factors are used, then the generator will yield again from the first (smallest) factor.

```
def generate_factors(k):
    """
    >> gen = generate_factors(4)
    >> next(gen)
    1
    >> next(gen)
    2
    >> next(gen)
    4
    >> next(gen)
    1
    >> next(gen)
    2
    >> next(gen)
    4
    >> next(gen)
    1
    """

    i = ____________________________________________________

    ____________________________________________________:

        if i _____________ k:

            i = ____________________________________________

        elif ____________ % _____________ == _____________:

            ____________________________________________

        i += ____________________________________________
```

(b) (2.0 pt) What would the output be if we called `list(generate_factors(10))`? Please note that we are using `list` and not `next`.

- ◯ `RecursionError: maximum recursion depth exceeded` error.
- ◯ `StopIteration` error.
- ◯ `[1, 2, 5, 10]`
- ◯ `[1, 2, 5, 10, 1, 2, ...]`
- ◯ An infinite loop

10. (4.0 points) May I take your order?

Recall, *orders of growth* allow us describe the efficency of a function, by explaining how much longer a function takes to run as its input size grows. In order from "fastest" to "slowest", they are:

- $O(1)$ — Constant time
- $O(log(n))$ — Logarithmic time
- $O(n)$ — Linear time
- $O(n^2)$ — Quadratic (or Polynomial) time
- $O(2^n)$ — Exponential time

Consider the following functions. Each is pretty short and accomplishes the same task.

```
def sum_nums_A(n):
    """
    >>> sum_nums_A(10)
    55
    """
    result = 0
    for i in range(n):
        for j in range(0, i + 1):
            result = result + 1
    return result
```

```
def sum_nums_B(n):
    """
    >>> sum_nums_B(10)
    55
    """
    return (n ** 2 + n ) // 2

def sum_nums_C(n):
    """
    >>> sum_nums_C(10)
    55
    """
    if n < 2:
        return n
    return n + sum_nums_C(n - 1)
```

(a) (1.0 pt) What is the order of growth of `sum_nums_A`?

- ◯ $O(1)$
- ◯ $O(log(n))$
- ◯ $O(n)$
- ◯ $O(n^2)$
- ◯ $O(2^n)$

(b) (1.0 pt) `sum_nums_A` will perform faster than `sum_nums_B` for large inputs

- ◯ `True`
- ◯ `False`

(c) (1.0 pt) `sum_nums_B` will perform faster than `sum_nums_C` for large inputs

- ◯ `True`
- ◯ `False`

(d) (1.0 pt) `sum_nums_C` will perform faster than `sum_nums_A` for large inputs

- ◯ `True`
- ◯ `False`

11. (14.0 points) NBA Networking

The female-identifying members on Data C88C staff are having a Girl's Night by networking with some NBA players interested in dabbling in data science after retirement from the league. They are trying to plan a hangout but don't know much about each other so everyone invited was asked to fill out "About Me" forms. Below are the two tables showcasing their responses: `staff` for course staff members, `players` for NBA players.

Tabble Name: `staff`

name	number	frequent_word	food_order	fav_player
Rebecca	88	bruh	rice	Kawhi Leonard
Jessica	64	slay	rice	Lebron James
Miha	21	slay	chicken	Kawhi Leonard
Christy	30	bruh	soup	Stephen Curry
Morgan	22	slay	beef	Luka Doncic
Ramya	8	bruh	tofu	Stephen Curry
Lily	21	slay	chicken	Lebron James
Angela	39	bruh	rice	Kevin Durant
Michelle	54	slay	chicken	Luka Doncic
Ananyaa	3	slay	tofu	Kawhi Leonard

Table Name: `players`

name	team	juice	food_order
Stephen Curry	Warriors	apple	soup
Lebron James	Lakers	orange	chicken
Kevin Durant	Suns	grape	beef
Kawhi Leonard	Clippers	apple	tofu
Luka Doncic	Mavericks	guava	chicken

(a) (4.0 pt) Write a query that selects the `name`s of staff members whose favorite number is greater than 23 grouped by their food order in alphabetical order (ascending). Running this query should return a table that looks like:

name
Michelle
Rebecca
Jessica
Angela
Christy

```
SELECT ____________________________________________________________

FROM ____________________________________________________________

WHERE ____________________________________________________________

GROUP BY ____________________________________________________________

ORDER BY ____________________________________________________________;
```

(b) (3.0 pt) Write a query that outputs the staff member's `name` and their favorite player's name `fav_player` where they have the same food order. The rows should be ordered by the staff members' names in alphabetical order (ascending). Running this query should return a table that looks like:

name	fav_player
Ananyaa	Kawhi Leonard
Christy	Stephen Curry
Lily	Lebron James
Michelle	Luka Doncic

```
SELECT ____________________________________________________________

FROM ______________________________________________________________

___________________________________________________________________

ORDER BY ___________________________________________________________;
```

(c) (3.0 pt) Write a query that outputs the `name` and `food_order` of all staff members whose favorite player prefers apple juice. Running this query should return a table that looks like:

name	food_order
Rebecca	rice
Miha	chicken
Christy	soup
Ramya	tofu
Ananyaa	tofu

```
SELECT ____________________________________________________________

FROM ______________________________________________________________

WHERE _____________________________________________________________

___________________________________________________________________;
```

(d) (4.0 pt) We want to find out what teams are most popular among C88C staff. Write a query that outputs the team and the total number of staff members as `total_staff` whose favorite player is on each team. The rows should be ordered by the values in `total_staff` numerically descending. Running this query should return a table that looks like:

team	total_staff
Clippers	3
Lakers	2
Mavericks	2
Warriors	2
Suns	1

```
SELECT ____________________________________________________________

FROM ______________________________________________________________

WHERE _____________________________________________________________

GROUP BY __________________________________________________________

ORDER BY _________________________________________________________;
```

12. (0.0 points) Bonus Questions!

(a) (1.0 pt) This is extra credit. Do not attempt it until you are done! Suggest a name for the CS88 Ant! (Any reasonable name counts for credit. :))

(b) (1.0 pt) Consider the following `fibonnaci` function to answer the following two questions:

```
fib_results = { 0: 0, 1: 1 }
def fib(n):
    """
    >>> 5 in fib_results
    False
    >>> fib(5)
    5
    >>> 5 in fib_results
    True
    """
    if n in fib_results:
        return fib_results[n]
    if n < 2:
        return n
    result = fib(n - 1) + fib(n - 2)
    fib_results[n] = result
    return result
```

This questions is extra credit. Do not attempt it until you are done! What is the order of growth of `fib`?

(c) (2.0 pt) This is extra credit. Do not attempt it until you are done! Draw the function calls for `fib(5)` as a tree. e.g. `fib(5)` should be the root node. Some inputs will be left empty.