

**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	
<i>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)</i>	

**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. (8.0 points) **Conceptual Questions**

Please note that each part is ran independently of each other.

(a) (1.0 pt) Suppose the following code has been executed:

```
x = [1, 2, 3]
y = [4, 5, 6]
x.append(y.pop(2))
x.remove(3)
```

What is the value of x now?

[1, 2, 6]

(b) (2.0 pt) Suppose the following code has been executed:

```
x = []
x is [] # This returns False
y = x
y.append(88)
```

Which of the following return True? (Select all that that apply.)

- x is y
- y is [88]
- x == y
- x == []

(c) (2.0 pt)

```
# Cat Class
class Cat:
    def __init__(self, size, age):
        self.size = size
        self.age = age
    def meow(self):
        print('meow')
```

```
# Kitten Class
class Kitten(Cat):
    #Implementation Hidden
```

Sally wants to use inheritance to help her create the Kitten class. Does the following code for Kitten's **init** method correctly use inheritance?

```
def __init__(self, age):
    super().__init__(age)
```

- No, the code will error because the Kitten class does not properly inherit from the Cat class
- Yes, the code uses inheritance to succesfully create the init method.
- No, the code will error because of a missing argument

(d) (2.0 pt) Now Sally wants to implement the `meow` method for the `Kitten` class. Suppose that the `Kitten` class no longer inherits from the `Cat` class. Additionally, we want a kitten to both meow and mew when the `meow` method is called. Does the code below accomplish this?

```
def meow(self):  
    Cat.meow(self)  
    print('mew')
```

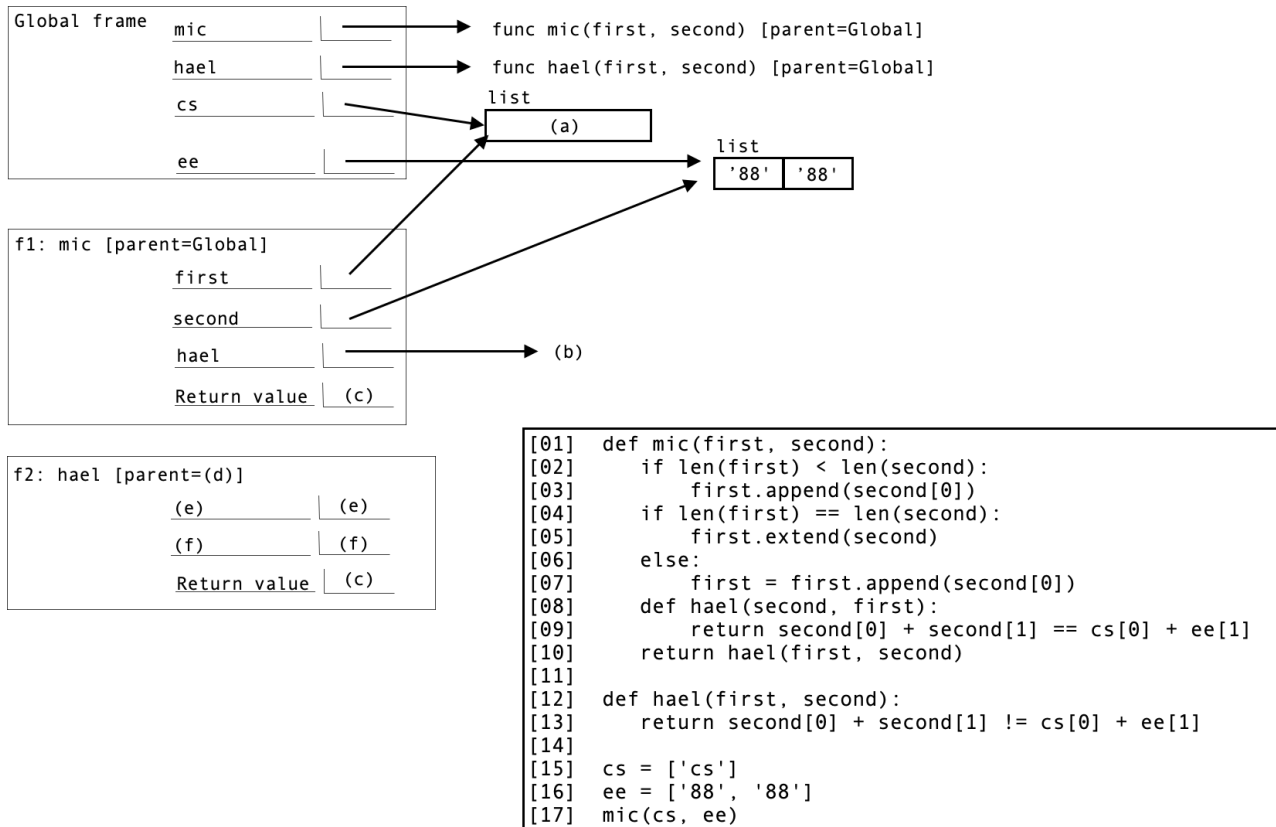
- No, the method does not get the kitten to both meow and mew
- Yes, this implementation of the `meow` method works correctly.
- No, the code does not properly call the `meow` method from the `Cat` class

(e) (1.0 pt) All branches of a tree are also themselves trees (i.e. they are all instances of the `Tree` class).

- False
- True

**2. (12.0 points) Michael Madness**

Fill in the environment diagram. For short answer questions, use relevant syntax and, if applicable, separate elements using commas. For example, if your answer is a list with integer elements 1, 2, and 3, write [1, 2, 3]. If your answer is 8.8, write 8.8. For string values, use single quotes, such as 'cs88'; do **not** use double quotes, such as "cs88". These rules will be **strictly** enforced: If your answer is correct but does not adhere to the above rules, you will **not receive any points** for it!



**Environment Diagram**

(a) (2.0 pt) Fill in the space denoted by (a). Remember to use the correct syntax when writing your answer.

['cs', '88', '88', '88']

(b) (2.0 pt) What is the function or object denoted by (b)?

- The list object denoted by (a)
- The list object ['88', '88']
- func hael(first, second) [parent=Global]
- func hael(second, first) [parent=f1]

(c) (2.0 pt) Fill in the spaces denoted by (c). Remember to use the correct syntax when writing your answer.

True

(d) (2.0 pt) What is the value denoted by (d)?

- Global
- f2
- f3
- f1

(e) (2.0 pt) What is the variable, pointer pair denoted by (e)?

- first, pointing to the list object denoted by (a)
- second, pointing to the list object denoted by (a)
- first, pointing to the list object ['88', '88']
- second, pointing to the list object ['88', '88']

(f) (2.0 pt) What is the variable, pointer pair denoted by (f)?

- first, pointing to the list object denoted by (a)
- first, pointing to the list object ['88', '88']
- second, pointing to the list object denoted by (a)
- second, pointing to the list object ['88', '88']

### 3. (8.0 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 executed/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 parts DO impact the current expression. (i.e., part B assumes part A was executed, as so on.) This means that variables that are previously defined/changed persist throughout the rest of the question

```
from functools import reduce
from operator import add, mul
```

```
func = lambda x: lambda: x
ingredients = {"flour": 5, "eggs": 2}
```

```
def stir(count, elements):
    res = []
    for i in range(count):
        for j, k in elements.items():
            if k % 2 == 0:
                res.append(func(k))
            else:
                res.extend([func(k)()])
    return res
```

```
adder = lambda x: [ i + 1 for i in x ]
```

(a) (1.0 pt)

```
>>> a = func([5,6])()
>>> a
```

[5, 6]

(b) (1.0 pt)

```
>>> b = adder(a)
>>> b
```

[6, 7]

(c) (1.0 pt)

```
>>> a = [1]
>>> print(a.extend([b]))
```

None

(d) (1.0 pt)

```
>>> a
```

[1, [6, 7]]

(e) (1.0 pt)

```
>>> c = stir(2, ingredients)
>>> c
```

- [5, Function]
- Error
- [5, 5, 5, 5]
- [5, 5]
- [5, Function, 5, Function]

(f) (1.0 pt)

```
>>> adder = lambda x: [ i + 1 for i in x ]
>>> list(map(lambda x: adder, range(3)))
```

- Error
- [None, None, None]
- [5, 5, 5]
- [Function, Function, Function]

(g) (1.0 pt)

```
>>> reduce(lambda x, y: x and y, [True, False, True])
```

- []
- Error
- False
- None
- True

(h) (1.0 pt)

```
>>> good_schools = filter(lambda s: s[0:2] == 'UC', ['UC Berkeley', 'Stanford'])
>>> next(good_schools)
```

- None
- Error
- ['UC Berkeley']
- 'Stanford'
- 'UC Berkeley'

#### 4. (12.0 points) Debugging

You're a new intern at a food review app company named Belly. You've been assigned to fix some mistakes that the past intern made while writing some important functions relating to generating food recommendations.

The previous intern wrote a series of functions that take in a dictionary of restaurants with the following structure:

```
>>> restaurants_and_ratings = {
    'Sliver': [2, 3.5, 5],
    'Cheeseboard': [1, 4],
    'Artichokes': [4]
}
```

Answer a series of questions that fix the previous intern's mistakes.

- (a) (4.0 pt) The first function that you're tasked to fix is one that takes in a *dictionary* where each key represents a *restaurant* and each value represents a list of that restaurant's ratings [*rating1*, *rating2*, ...]. It should output a list, where each item is a 2-item list [*restaurant*, *average\_rating*]. Here is the function:

```
def generate_list(d):
    avgs = []
    for restaurant in d:
        avg = sum(d) / len(d)
        avgs.append(restaurant, avg)
    return avgs
```

The intended output is as follows:

```
>>> restaurants_and_ratings = {
    'Sliver': [2, 3.5, 5],
    'Cheeseboard': [1, 4],
    'Artichokes': [4]
}
>>> generate_list(restaurants_and_ratings)
[['Sliver', 3.5], ['Cheeseboard', 2.5], ['Artichokes', 4.0]]
```

Unfortunately, the current code has some bugs. How would you fix this function so the code does not error and calling the function results in the correct output? (Please Select All that apply. Multiple choices may need to be selected for the code to run correctly.)

- Replace `sum(d) / len(d)` with `sum(d[restaurant]) / len(d[restaurant])`
- Replace `d` with `d.items()`.
- Replace `avgs.append(restaurant, avg)` with `avgs.append([restaurant, avg])`
- Replace `avgs.append(restaurant, avg)` with `avgs.extend(restaurant, avg)`



- (b) (4.0 pt) The next function that you're tasked to fix is one that takes in the generated *list* of restaurants and their average ratings and outputs a list, sorted from highest to lowest average rating. Here it is:

```
def sort_restaurants(l):
    if len(l) <= 1:
        return l
    else:
        first = l[0]
        lst1 = [e for e in l[1:] if e[1] <= l[0][1]]
        lst2 = [e for e in l[1:] if e[1] > l[0][1]]
        return sort_restaurants(lst1) + [first] + sort_restaurants(lst2)
```

Here is the intended output:

```
>>> restaurant_list = [['Sliver', 3.5], ['Cheeseboard', 3.0], ['Artichokes', 4.0]]
>>> sort_restaurants(restaurant_list)
[['Artichokes', 4.0], ['Sliver', 3.5], ['Cheeseboard', 3.0]]
```

Will the previous intern's code result in the correct output *and* run without erroring? Select *only the option* that has the correct combination of Yes/No.

- No, this code will order the list incorrectly (ascending). Yes, this code will not error.
  - Yes, this code will order the list correctly (descending). Yes, this code will not error.
  - Yes, this code will order the list correctly (descending). No, this code will error; `l[0][1]` should be replaced with `l[0]`.
  - No, this code will order the list incorrectly (ascending). No, this code will error; `len(l) <= 1` should be replaced with `len(l) == 0`.
- (c) (4.0 pt) You can now assume that `sort_restaurants` results in the correctly ordered list. To save memory in the company's database, your manager asks you to modify `sort_restaurants` so that it *mutates the list passed in*. This means that the updated intended output will be as following:

```
>>> restaurant_list = [['Sliver', 3.5], ['Cheeseboard', 3.0], ['Artichokes', 4.0]]
>>> sort_restaurants(restaurant_list)
>>> restaurant_list
[['Artichokes', 4.0], ['Sliver', 3.5], ['Cheeseboard', 3.0]]
```

Will you need to modify `sort_restaurants` further? Select *only the option* that has the correct combination of Yes/No and reasoning.

- No, the current code mutates the list passed in.
- Yes, but we cannot use recursion because it is not possible to mutate a list using recursion.
- Yes, because list slicing creates a copy of the given list and doesn't mutate it directly.
- Yes, and we need to use list functions such as `.append()`, `.extend()`, and `pop()`.

### 5. (11.0 points) Backyard Zoo

Michelle is trying to organize summer gift baskets for her backyard monkeys, but they each require different supplies! Help her implement the class `Monkey` to help her decide how many m&ms, silly bands, and ring pops to put in the gift baskets for each of these monkeys in her zoo.

```
class Monkey(LandAnimal):
    def __init__(self, name, species, weight):
        # part (a)
    def count_silly_bands(self):
        # part (b)
    def items_in_gift_basket(self):
        # part (c)
```

(a) (3.0 pt) Below is the implementation for the `LandAnimal` class

```
class LandAnimal():
    def __init__(self, species, weight):
        self.species = species
        self.weight = weight
        self.ring_pops = 3

    def count_m_and_ms(self):
        """return how many m&ms to put in a land animal's basket"""
        return 5 * self.weight
```

Now since Monkeys are land animals, fill in the blanks to make the constructor of the `Monkey` class inherit from the `LandAnimal` class. In addition to what's initialized for every `LandAnimal`, we have a `name` for each monkey!

```
class Monkey(LandAnimal):
    def __init__(self, name, species, weight):
        """
        >>> george = Monkey("george", "curious monkey", 10)
        >>> george.name
        'george'
        >>> george.species
        'curious monkey'
        >>> george.weight
        10
        """
        ...
```

```
class Monkey(LandAnimal):
    def __init__(self, name, species, weight):
        super().__init__(species, weight)
        self.name = name
```

- (b) (3.0 pt) To count the number of silly bands to put in each monkey's basket, Michelle came up with a formula to optimize her pet monkeys' happiness. All of her monkeys start off with 10 silly bands. She then adds a silly band for each even number between 1 (inclusive) and the length of the monkey's name (exclusive). For some examples, look at the doctests below.

You may assume the monkey does not have an empty string for a name.

```
def count_silly_bands(self):
    """
    >>> monkey1 = Monkey("bob", "monkey", 12)
    >>> monkey1.count_silly_bands()
        # bob is of length 3, which only contains 1 even value (2),
        # so only one silly band is added
    11 # 10 + 1 = 11
    >>> monkey2 = Monkey("stephanie", "monkey", 12)
    >>> monkey2.count_silly_bands()
        # "stephanie" is of length 9, which contains 4 even values
        # (2, 4, 6, 8), so 4 silly bands are added
    14 # 10 + 4 = 14
    """
    ...
```

```
def count_silly_bands(self):
    res = 10
    for i in range(1, len(self.name)):
        if i % 2 == 0:
            res += 1
    return res
```

- (c) (3.0 pt) Now we want to implement the function `items_in_gift_basket` which will return a three element list that represents how many associated m&ms, silly bands, and ring pops to put in each basket respectively.

```
def items_in_gift_basket(self):
    return [self.count_m_and_ms(), self.count_silly_bands(), self.ring_pops]
```

- (d) (2.0 pt) Michelle has now decided that all the Monkeys that are of the species 'curious monkey' deserve an additional 3 ring pops, but only if the Monkey's name is longer than 5 letters. Which of these actions should Michelle take to adapt her code? (The change should not break any of the other parts of this question.)

- Modify the constructor method of the `Monkey` class
- Add a new method `count_ring_pops` to the `Monkey` class
- Modify the constructor method of the `LandAnimal` class
- Add a new method `count_ring_pops` to the `LandAnimal` class

### 6. (12.0 points) Linked List Chains

We want to create a linked list chain with length  $n$ . Given an input list called `lst`, we will go through each element one by one and add each to our new resulting linked list. If there are no more elements left in the input list before the result linked list reaches length  $n$ , go back to the beginning of the input list to use the elements in order again. Assume that calling `print` on any linked list will format the values in order, enclosed by angle brackets `<` and `>` (you will not need to implement this yourself).

```
def link_chain(lst, n):
    """
    >>> res_lnk = link_chain(['o_0'], 3)
    >>> res_lnk
    Link('o_0', Link('o_0', Link('o_0')))
    >>> print(res_lnk)
    <'o_0', 'o_0', 'o_0'>
    >>> print(link_chain([1, 2], 7))
    <1, 2, 1, 2, 1, 2, 1>
    >>> print(link_chain([], 2))
    <>
    """
    if _____(a)_____ or _____(b)_____:
        return _____(c)_____
    lnk_rest = link_chain(_____ (d) _____, _____(e)_____)
    lnk_first = _____(f)_____
    return Link(lnk_first, lnk_rest)
```

(a) (2.0 pt) Fill in blank (a). Select all options that would correctly fill in the blank.

- `lst` is `List.empty`
- `lst == []`
- `lst` is `Link.empty`
- `not lst`
- `lst` is `[]`

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

`n == 0`

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

`Link.empty`

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

`lst[1:] + [lst[0]]`

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

- `n`
- `n-1`
- `len(lst) - n`
- `len(lst)`

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

1st[0]

7. (10.0 points) Trees

Implement `sum_depth_k`, which takes in a `Tree` instance and returns the sum of the values at depth `k`. Assume that `k` will always be less than or equal to the height of the tree.

```
def sum_depth_k(t, depth):  
    """  
    >>> t = Tree(1, [Tree(2, [Tree(4), Tree(5)]), Tree(6, [Tree(7)])])  
    >>> sum_depth_k(t, 0)  
    1  
    >>> sum_depth_k(t, 1) # 2 + 6  
    8  
    >>> sum_depth_k(t, 2) # 4 + 5 + 7  
    16  
    """  
    if depth == 0:  
        return ____ (a) ____  
    elif t.is_leaf():  
        return ____ (b) ____  
    else:  
        total = ____ (c) ____  
        for ____ (d) ____:  
            ____ (e) ____  
        return total
```

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

```
return t.value
```

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

```
return 0
```

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

```
total = 0
```

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

```
for b in t.branches
```

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

```
total += sum_depth_k(b, depth - 1)
```

8. (11.0 points) Cyclic Generator

Implement the generator function `cyclic_generator` which takes in a non-empty list of one-argument functions `fns` and a non-empty list `values`, and applies the functions to the elements of `values` in a cyclic fashion **infinitely**. Note that `len(fns)` is **NOT necessarily the same** as `len(values)`. Read the doctests for examples.

```
def cyclic_generator(fns, values):
    """
    >>> f1 = lambda x: x
    >>> f2 = lambda x: x + 1
    >>> f3 = lambda x: 10 - x
    >>> fns = [f1, f2, f3]
    >>> values = [2, 3, 4]
    >>> gen = cyclic_generator(fns, values)
    >>> next(gen) # f1(2) = 2
    2
    >>> next(gen) # f2(3) = 4
    4
    >>> next(gen) # f3(4) = 6
    6
    >>> next(gen) # cycle repeats infinitely afterward
    2
    >>> values = [2, 3, 4, 5] # len(values) != len(fns)
    >>> gen = cyclic_generator(fns, values)
    >>> next(gen) # f1(2) = 2
    2
    >>> next(gen) # f2(3) = 4
    4
    >>> next(gen) # f3(4) = 6
    6
    >>> next(gen) # f1(5) = 5
    5
    >>> next(gen) # f2(2) = 3
    3
    >>> next(gen) # f3(3) = 7
    7
    >>> next(gen) # f1(4) = 4
    4
    """
    fns_index = 0
    values_index = 0
    while ___(a)___:
        curr_fn = ___(b)___
        curr_val = ___(c)___
        ___(d)___ curr_fn(curr_val)
        fns_index += ___(e)___
        values_index += ___(e)___
```

Hint: Use the modulo (%) operator. To see a special property of modulo, what do you notice when you do  $0 \% 3$ ,  $1 \% 3$ ,  $2 \% 3$ ,  $3 \% 3$ ,  $4 \% 3$ ,  $5 \% 3$ ,  $6 \% 3$ , etc.? How can you generalize this idea to achieve the cyclic nature of the function? If you recall the `cycle` problem from HW 4, this problem is very similar!

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

- True
- `values_index < len(values)`
- `fns_index < len(fns)`
- False

(b) (3.0 pt) What should fill in blank (b)?

```
fns[fns_index % len(fns)]
```

(c) (3.0 pt) What should fill in blank (c)?

```
values[values_index % len(values)]
```

(d) (1.0 pt) What should fill in blank (d)?

```
yield
```

(e) (2.0 pt) What should fill in blank (e)? **Note that blank (e) occurs twice - both have the same value.**

```
1
```



9. (5.0 points) Efficient or Not

(a) (1.0 pt) What is the run time of the following function `counter_squared`? Let  $n$  be the length of `lst`.

```
def counter_squared(lst):  
    counter = 0  
    new_lst = []  
    for i in lst:  
        counter *= i * i  
        new_lst.append(counter)  
    return new_lst
```

- $O(1)$
- $O(\log(n))$
- $O(n)$
- $O(n\log(n))$
- $O(n^2)$
- $O(2^n)$
- None of the above

(b) (1.0 pt) What is the run time of the following function `random_func`?

*Hint:* you don't need to calculate the exact runtime.

```
def random_func(n):  
    val = 0  
    for i in range(n):  
        for j in range(i):  
            val += sum(map(lambda x: x * x, range(n)))  
    return val/n
```

- $O(1)$
- $O(\log(n))$
- $O(n)$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$
- None of the above

(c) (1.0 pt) Let us say that we create a new function `random_func_new` that is the same as `random_func` except the 5th line has been changed from `val += sum(...)` to `val += (n - j) ** 2`.

How will the run time of the new function change in relation to the original function in part b?

The new function is written here for clarity:

```
def random_func_new(n):  
    val = 0  
    for i in range(n):  
        for j in range(i):  
            val += (n - j) ** 2  
    return val/n
```

- `random_func_new` is more efficient than `random_func`.
- `random_func_new` is equally as efficient as `random_func`.
- `random_func_new` is less efficient than `random_func`.

(d) (1.0 pt) What is the run time of the following function `weird_func`?

```
def weird_func(n):  
    val = 0  
    while n > 0:  
        val += n  
        val //= 2  
        n //= 2  
    return val
```

- $O(1)$
- $O(\log(n))$
- $O(n\log(n))$
- $O(n)$
- $O(n^2)$
- $O(2^n)$
- None of the above

(e) (1.0 pt) What is the run time of the following function `maps_on_maps_maps`?

```
def maps_on_maps_maps(n):  
    return list(map(lambda x: x ** x, filter(lambda y: y % 2 == 0, range(n))))
```

- $O(1)$
- $O(\log(n))$
- $O(n\log(n))$
- $O(n)$
- $O(n^2)$
- $O(2^n)$
- None of the above

10. (15.0 points) C88C Game Night

The members of the Data C88C staff are organizing a Game Night to unwind after a long semester. To ensure everyone enjoys the event, all members were asked to fill out a “Game Preferences” form. Below are the two tables showcasing their responses: `staff` for the course staff members and `games` for information about the games.

Table Name: `staff`

name	drink_pref	msg_pref	snack_pref	fav_game
Ethan	soda	text	chips	Mario Party
Rebecca	coffee	slack	cookies	Mario Party
Angela	tea	text	fruit	Jackbox
John	soda	email	fruit	Among Us
Karim	water	messenger	chips	Valorant
Lily	coffee	text	cookies	Stardew Valley
Michelle	coffee	email	chips	Minecraft
Ramya	tea	slack	fruit	Stardew Valley
Sean	water	instagram	fruit	Valorant
Ethan	water	slack	cookies	League
Miha	soda	text	cookies	Mario Party
Morgan	soda	messenger	chips	Mario Party
Nicholas	soda	text	cookies	League
Richik	soda	instagram	fruit	Minecraft
Satleen	water	text	chips	Minecraft

Table Name: `games`

name	company	genre	max_players	
Mario Party		Nintendo	Party	4
Jackbox	Jackbox Games		Party	10
Among Us		Innersloth	Party	15
Valorant		Riot	FPS	5
Stardew Valley	ConcernedApe		Indie	4
Minecraft		Mojang	Indie	40
League		Riot	Moba	5

(a) (3.0 pt) Write a query that outputs the `name` and `msg_pref` of all staff members whose favorite game is made by Riot. Running this query should return a table that looks like:

name	msg_pref
Karim	messenger
Sean	instagram
Ethan	slack
Nicholas	text

```
SELECT staff.name, msg_pref
FROM staff, games
WHERE staff.fav_game = games.name AND games.company = 'Riot';
```

- (b) (3.0 pt) Write a query to list the **names** of staff members who prefer games that can accommodate 10 or more players, along with their snack preferences. Order the results alphabetically (ascending) by the staff member's name. Running this query should return a table that looks like:

name	snack_pref
Angela	fruit
John	fruit
Michelle	chips
Richik	fruit
Satleen	chips

```
SELECT staff.name, snack_pref
FROM staff, games
WHERE fav_game = games.name AND max_players >= 10
ORDER BY staff.name ASC;
```

- (c) (4.0 pt) We want to find out what genres of games are the most popular among the C88C staff. Write a query that outputs the **genre** and the total number of staff that prefers the genre. We want to list the genres in descending order by the number of staff. Break ties by alphabetical order of the genre (ascending). Running this query should return a table that looks like:

genre	total_staff
Party	6
Indie	5
FPS	2
Moba	2

```
SELECT genre, COUNT(*) AS total_staff
FROM staff
JOIN games
ON staff.fav_game = games.name
GROUP BY genre
ORDER BY total_staff DESC, genre ASC;
```

- (d) (5.0 pt) Write a query to count how many staff members prefer each game, grouped by their drink preference. Display the **fav\_game**, the **drink\_pref**, and the count of staff members. Only include rows where the total count of **fav\_game** is more than 1 for that particular drink. Order the results by the number of staff members in descending order. Running this query should return a table that looks like:

fav_game	drink_pref	staff_count
Mario Party	soda	3
Valorant	water	2

```
SELECT fav_game, drink_pref, COUNT(*) as staff_count
FROM staff
GROUP BY fav_game, drink_pref
HAVING COUNT(*) > 1
ORDER BY staff_count DESC;
```

**11. (0.0 points) Bonus Question!**

This question is a bonus question, so you will not lose points if leave it blank. Take your best guess (the question was deliberately written as is and there are no errors in it)! This entire page may be worth up to 2 points in total.

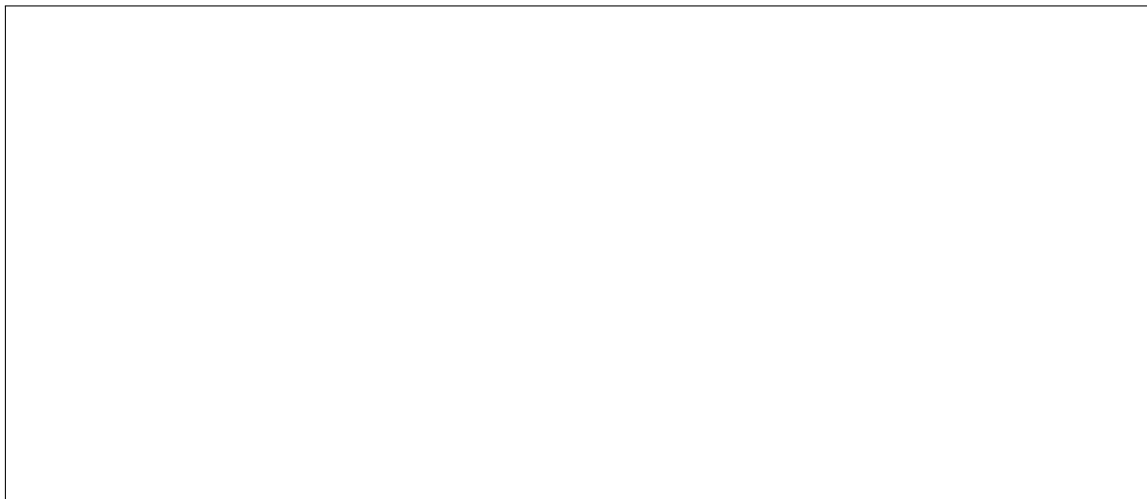
- (a) (0.0 pt) The instructor of this class, Michael Ball, was Head TA for CS10 during his undergrad at Berkeley. In what year did he win the “Outstanding GSI Award”? For reference, he won the “Outstanding GSI Award” in 2015.

2015

- (b) (0.0 pt) What is the value of `True ** False` in Python? Believe it or not, this is totally valid Python, even though we think it’s a silly feature.

1

- (c) (0.0 pt) Thanks for a great semester! Draw something fun!



<NAME> SID: \_\_\_\_\_

**No more questions.**