CS 88 Spring 2023 — Computational Structures in Data Science

MIDTERM

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)	<EMAILADDRESS>
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.
  - **You must include all answers within the boxes.**
  - **Online Exams: You may start your exam as soon as you are given the password.**
    - **You may have a digital version of the C88C Reference Sheet, but no other files.**
    - Exam Clarifications: https://tinyurl.com/clarifications-sp23
    - Reference Sheet: https://tinyurl.com/mt-reference

1. (3.0 points) Conceptual

(a) (0.5 pt) `{"2M": "Rent", "2M": "It's My Birthday"}` is a valid dictionary.

○ `True`

○ `False`

(b) (0.5 pt) `{"Sly": "Deal", "Forced": "Deal"}` is a valid dictionary.

○ `True`

○ `False`

(c) (1.0 pt) Which function call to `monopoly_deal` would result in an infinite loop?

```
def monopoly_deal(num_cards, pass_go):
    num_cards += 2
    while num_cards > 7:
        if pass_go:
            num_cards += 2
        else:
            num_cards -= 1
            if num_cards % 2 == 0:
                print("debt collector")
            else:
                print("deal breaker")
    return None
```

○ `monopoly_deal(0, True)`

○ `monopoly_deal(7, True)`

○ `monopoly_deal(0, False)`

○ `monopoly_deal(7, False)`

(d) (0.5 pt) Below is the code for the Player ADT

```
def make_player(name, game):
    return {"name": name, "game": game}

def get_player_name(player):
    return player["name"]

def get_player_game(player):
    return player["game"]
```

Does the following code break the Abstraction Barrier?

```
aymeric = make_player("Aymeric Barrier", "Monopoly Deal")
print(get_player_name(aymeric) + " lost at " + aymeric["game"])
```

○ `Breaks Abstraction Barrier`

○ `Does NOT Break Abstraction Barrier`

(e) (0.5 pt) Does the following code break the Abstraction Barrier?

```
hetal = make_player("Hetal Shah", "Monopoly Deal")
print(get_player_name(hetal) + " won at " + get_player_game(hetal))
```

○ `Breaks Abstraction Barrier`

○ `Does NOT Break Abstraction Barrier`

2. (3.0 points) Mutation and Creation Conceptual Questions

(a) (0.5 pt) Based on the given example output, does the function `mystery1` mutate the input list or return a new list?

```
>>> lst = [1, 2, 3]
>>> mystery1(lst)
>>> lst
[2, 3, 4]
```

○ `mystery1` mutates the input list

○ `mystery1` returns a new list

(b) (0.5 pt) Based on the given example output, does the function `mystery2` mutate the input list or return a new list?

```
>>> lst = [1, 2, 3]
>>> mystery2(lst)
[2, 3, 4]
>>> lst
[1, 2, 3]
```

○ `mystery2` mutates the input list

○ `mystery2` returns a new list

(c) (1.0 pt) You are given the following function:

```
def remove_threes(lst):
    return list(filter(lambda x: x != 3, lst))
```

You are also given the list `input = [1, 2, 3]`.

Will the function call `remove_threes(input)` mutate `input`?

○ `input` will be mutated

○ `input` will not be mutated

(d) (1.0 pt) You are given the following function:

```
def duplicate_lst(lst):
    new_lst = lst
    for x in lst:
        new_lst.append(x)
    return new_lst
```

You are also given the list `input = [1, 2, 3]`.

Will the function call `duplicate_lst(input)` mutate `input`?

○ `input` will be mutated

○ `input` will not be mutated

3. (5.0 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 the expression evaluates to a function, write “Function”.

```
>>> x = 5
>>> y = 4
>>> z = [lambda y: y, lambda x: x, lambda x: y]
>>> def g(l):
...     x = 6
...     return l(x)
...
>>> r = lambda n: lambda phi: z[-2](phi(n))
```

(a) (1.0 pt)

```
>>> z[1]
```

(b) (1.0 pt)

```
>>> [q(x) for q in z]
```

(c) (1.0 pt)

```
>>> [x + n for n in z]
```

(d) (1.0 pt)

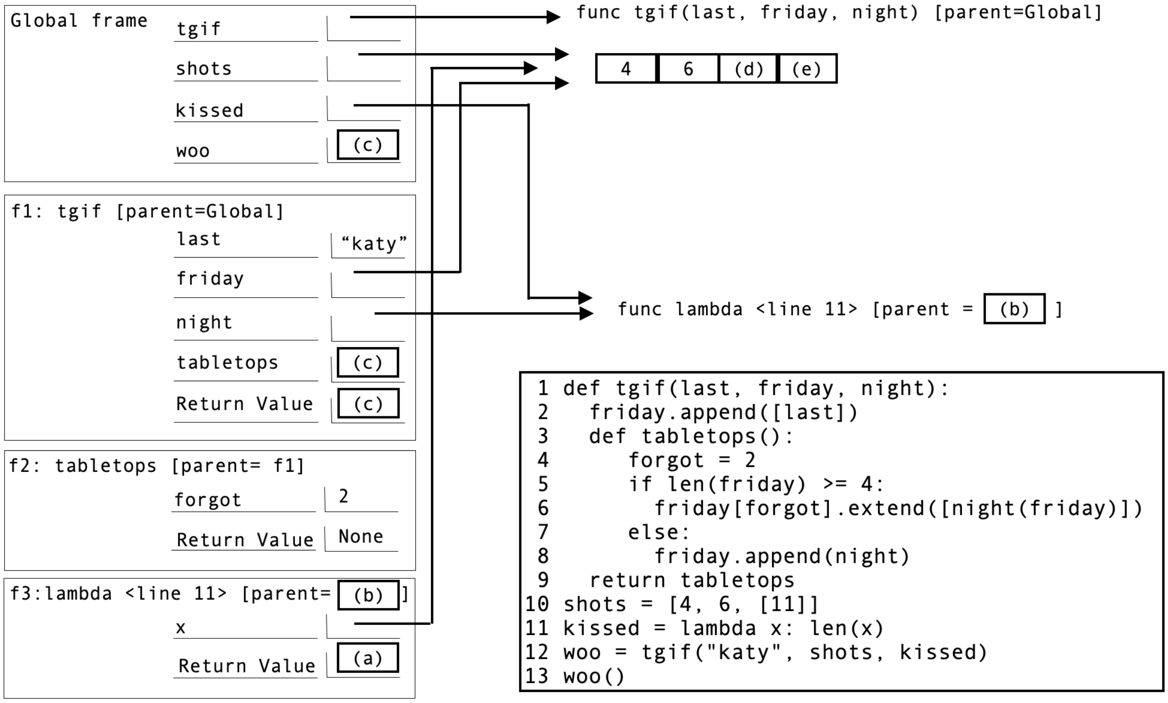
```
>>> g(lambda y: x + 1)
```

(e) (1.0 pt)

```
>>> r(1)(2)
```

4. (5.0 points) TGIF!

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.



```
 1 def tgif(last, friday, night):
 2   friday.append([last])
 3   def tabletops():
 4      forgot = 2
 5      if len(friday) >= 4:
 6        friday[forgot].extend([night(friday)])
 7      else:
 8        friday.append(night)
 9   return tabletops
10 shots = [4, 6, [11]]
11 kissed = lambda x: len(x)
12 woo = tgif("katy", shots, kissed)
13 woo()
```

(a) (1.0 pt) What is the return value of the lambda function in frame `f3`? (box a)

(b) (1.0 pt) What is the parent frame of the lambda function in frame `f3`? (box b)

(c) (1.0 pt) What is the value of `woo`, `tabletops` and the return value of the `f1` frame? (box c)

Note: these three values point to the same thing!

○ `func tabletops`

○ `shots`

○ `func lambda <line 11>`

○ `func tgif`

(d) (1.0 pt) What is the element at index 2 in `shots` ? (box d)

(e) (1.0 pt) What is the element at index 3 in `shots` ? (box e)

5. (3.0 points) Debugging

Your friend Alice is trying to write a function `count_elements` that counts how many times a number appears in a list. This function returns a dictionary where each number is mapped to its count.

```
>>> lst = [1, 4, 5, 4, 7, 7, 2, 7]
>>> count_elements(lst)
{1: 1,                  # 1 appears once
4: 2,                   # 4 appears twice
5: 1,
7: 3,                   # 7 appears three times
2: 1}
```

Here is Alice's code:

```
def count_elements(lst):
    alice_dict = {}
    for elem in lst:
        alice_dict[elem] += 1
    return alice_dict
```

Use this code to answer the following question.

(a) (1.0 pt) Will this code execute the expected behavior?

○ Yes, this code correctly counts how many times each element appears in the list and stores it in `alice_dict`.

○ No, this code's count is less than the actual count by 1 (i.e. the code counts 2 when the element appears 3 times).

○ No, this code errors when it reaches an element of the list that already appeared previously.

○ No, this code always errors on the first iteration of the `for` loop.

○ No, this code is incorrect for a different reason not listed here.

(b) **(2.0 pt)** Your friend Bob tries to do something similar using his function `bob_func`. However, instead of storing how many times the number appears as the value, he wants to store a list of indices at which the number appears.

```
>>> lst = [1, 4, 5, 4, 7, 7, 2, 7]
>>> bob_func(lst)
{1: [0], # 1 appears at index 0
4: [1, 3], # 4 appears at indices 1 and 3
5: [2],
7: [4, 5, 7],
2: [6]}
```

Here is Bob's code:

```
def bob_func(lst):
    bob_dict = {}              # line 1
    for i in lst:              # line 2
        elem = lst[i]          # line 3
        bob_dict[elem] = i     # line 4
    return bob_dict            # line 5
```

Here is Bob's explanation for his code:

> "First, I create a new empty dictionary to hold my result. Then, I iterate over the indices of the list since I know that I need to keep track of which index each element corresponds to. On line 3, I define a new variable `elem` and assign it to `lst[i]`, which is the element of the list that the loop is currently on. On the next line, I update my dictionary using `elem` as the key and the index, `i`, as the value. At the end, I return the dictionary that I created."

But, Bob's code does not exactly match his explanation, and he also does not fulfil the problem's original goal. Which of the following suggestions should Bob implement to make his code more closely match his explanation and the problem's goal? **(Select all that apply)**

□ On line 2, change the `for` loop to `for i in range(len(lst)):`

□ On line 2, change the `for` loop to `while i in lst:`

□ Replace line 3 with `elem = i`.

□ Replace line 4 with `bob_dict[elem].append(i)`.

□ Replace line 4 with `bob_dict[elem].extend(i)`.

6. (7.0 points) Sum to Single Digit

(a) (7.0 pt) Given a number `num`, find the sum of all the digits of `num`. If this sum is not a single digit number, find the sum of all *its* digits. Continue this process until the sum has only one digit.

Print each sum including the last one digit sum you get, and return how many times you went through this process.

Note: `num` is always nonnegative.

```
def sum_to_single_digit(num):
    """
    >>> a = sum_to_single_digit(123) # num = 123, 1st sum = 6, terminate
    123
    6
    >>> a # had to sum digits once
    1
    >>> b = sum_to_single_digit(12345) # num = 12345, 1st sum = 15, 2nd sum = 6, terminate
    12345
    15
    6
    >>> b # had to sum digits twice
    2
    """
    count = _________________________________
    while _________________________________:
        _________________________________
        total = _________________________________
        while num _________________________________:
            total += _________________________________
            num = num // 10
        num = _________________________________
        _________________________________
    print(num)
    return _________________________________
```

```
def sum_to_single_digit(num):

    count = ____________________________________________________

    while _____________________________________________________:

        ________________________________________________________

        total = ________________________________________________

        while num _____________________________________________:

            total += ___________________________________________
            num = num // 10

        num = __________________________________________________

        ________________________________________________________
    print(num)

    return ____________________________________________________________
```

7. (6.0 points) C88C Voting Machine

(a) (2.0 pt) You are implementing the `add_votes` function for the C88C voting machine. This function takes in the `name` of the candidate, the `amount` of votes they recieved, and a dictionary `votes` which maps candidate names to their number of votes.

The function should modify `votes` by either updating the candidate's current entry or creating a new entry for the candidate.

```
def add_votes(name, amount, votes):
    """
    >>> votes = {'Michael': 400, 'Hetal': 100}
    >>> add_votes('Karim', 75, votes) // add new entry
    >>> votes = {'Michael': 400, 'Hetal': 100, 'Karim': 75}
    >>> add_votes('Michael', 100, votes) // update existing entry
    >>> votes = {'Michael': 500, 'Hetal': 100, 'Karim': 75}
    """
    if _____________________________________________:
        _____________________________________________
    else:
        _____________________________________________
```

```
def add_votes(name, amount, votes):

    if _____________________________________________:

        _____________________________________________
    else:

        _____________________________________________
```

(b) **(4.0 pt)** For this part, you will now have access to another dictionary `parties` that maps the name of a party to a list of candidate names within that party. Using the `parties` and `votes` dictionary, implement the `vote_summary` function which prints the name of each party followed by the total number of votes that all of the candidates within that party recieved.

```
def vote_summary(votes, parties):
    """
    >>> parties = {'Professor Party': ['Michael'], 'TA Party': ['Karim', 'Hetal']}
    >>> votes = {'Michael': 500, 'Hetal': 100, 'Karim': 75}
    >>> vote_summary(votes, parties)
    Professor Party : 500
    TA Party : 175
    """
    for ____________________________:
        count = ____________________________
        for name in votes:
            if ____________________________:
                ____________________________
        print( ____________________________, ':', count)
```

```
def vote_summary(votes, parties):

    for _______________________________________________________:

        count = ________________________________________________
        for name in votes:

            if ________________________________________________:

                ________________________________________________

        print(_____________________________________, ’:’, count)
```

8. (3.0 points) Higher Order Combiners

Implement `high_order_combiner`, which takes in a two-argument function `combiner`, an integer `start` representing the start value, and a one-argument function `is_brake` which returns either `True` or `False`.

When `high_order_combiner` is called, it will continuously return one-argument functions until the argument to said function is a brake, i.e., calling `is_brake` on the argument returns `True`. Once a brake argument is received, the high order combiner returns the result of combining all arguments passed in.

```
def high_order_combiner(combiner, start, is_brake):
    """
    >>> c1 = high_order_combiner(lambda x, y: x + y, 0, lambda x: x % 2 == 0)
    >>> c1(1)(3)(5)(6) # 0 + 1 + 3 + 5 + 6
    15
    >>> c2 = high_order_combiner(lambda x, y: max(x, y), 8, lambda x: x % 10 == x // 10)
    >>> c2(888)(88) # maximum of 8, 888, and 88
    888
    >>> c2(88)
    88
    >>> c2(8) # when a brake is not encountered yet, return another function
    <function <lambda> at ...>
    >>> c2(8)(88)(888) # c2(8)(88) returns an integer, so it cannot accept the additional argument 888
    TypeError: 'int' object is not callable
    """
    def helper(x, combo):
        if is_brake(x):
            return ____(a)____
        else:
            return lambda y: helper(y, ____(b)____)
    return ____(c)____
```

(a) **(1.0 pt)** Which of these could fill in blank (a)?

○ `combo`

○ `x`

○ `combiner(x, combo)`

○ `combiner(x, start)`

○ `combiner(start, combo)`

(b) **(1.0 pt)** Which of these could fill in blank (b)?

○ `combo`

○ `x`

○ `combiner(x, combo)`

○ `combiner(x, start)`

○ `combiner(start, combo)`

(c) **(1.0 pt)** Which of these could fill in blank (c)?

○ `helper`

○ `combiner`

○ `lambda x: helper(x, start)`

○ `lambda x: combiner(x, start)`

○ `lambda x: higher_order_combiner(combiner, x, is_brake)`

9. (6.0 points) Stocko Mode

In this problem, we will be implementing a simplified stock market using ADTs. In this market, people can buy and sell stocks at a set market rate. To implement this market, we will be using two ADTs, the Stock Market ADT and the Account ADT. The account holds the balance in someone's account as well as all of the stocks that they own. The market holds the prices for the various stocks.

Implement the Stock Market ADT and the Account ADT:

- The Stock Market ADT is internally represented as a **dictionary** which maps *tickers* to prices (an integer number). The abbreviated name of a stock is known as a *ticker*, e.g. "AAPL" for "Apple".
- The Account ADT is internally represented as a **list** with a length of 2. The first element is a list of all the stocks that the account owns while the second argument is an integer representing the balance of the account.

After this we will implement some client-side code that will utilize these ADTs to complete a transaction. **For the client-side code, remember to respect the Abstraction Barrier.**

```
# Stock Market ADT
def create_market():
    """
    >>> m = create_market()
    >>> add_or_update_price(m, "CS", 88)
    >>> johnnys_acct = make_account(100)
    >>> buy_stock(m, johnnys_acct, "CS")
    >>> get_assets(johnnys_acct)
    ["CS"]
    >>> add_or_update_price(m, "CS", 188)
    >>> sell_stock(m, johnnys_acct, "CS")
    >>> get_balance(johnnys_acct)
    200
    """
    return {}
def add_or_update_price(stock_market, ticker, price):
    _________________________________________
def get_price(stock_market, ticker): # assume that the stock exists in the market
    return __________________________________

# Account ADT
def make_account(starting_money):
    return [[], starting_money]
def add_money(account, amount):
    _____________________________________
def subtract_money(account, amount)
    add_money(account, -amount)
def add_stock(account, ticker):
    _____________________________________
def remove_stock(account, ticker): # assume the stock is in the portfolio
    account[0].remove(ticker)
def get_balance(account):
    return account[1]
def get_assets(account):
    return account[0]

# Client-side code
def buy_stock(stock_market, account, ticker): # assume the stock exists in the market
    ______________________________________
    ______________________________________

def sell_stock(stock_market, account, ticker):
    ______________________________________
    ______________________________________
```

(a) (1.0 pt) The following two questions ask you to complete functions in the Stock Market ADT. Implement the function `add_or_update_price`, which is part of the Stock Market ADT.

```
def add_or_update_price(stock_market, ticker, price):

    _________________________________________________
```

(b) (1.0 pt) Implement the function `get_price`, which is part of the Stock Market ADT.

```
def get_price(stock_market, ticker):

    ________________________________________
```

(c) (1.0 pt) The following two questions ask you to complete functions in the Account ADT. Implement the `add_money` function which is part of the Account ADT.

```
#Account ADT
def make_account(starting_money):
    return [[], starting_money]
def add_money(account, amount):
    _________________________________________________________
```

```
def add_money(account, amount):

    _________________________________________________________
```

(d) (1.0 pt) Implement the `add_stock` function which is part of the Account ADT.

```
def make_account(starting_money):
    return [[], starting_money]
def add_stock(account, ticker):
    _______________________________________________
def remove_stock(account, ticker):
    account[0].remove(ticker)
```

```
def add_stock(account, ticker):

    _________________________________________________________
```

(e) (1.0 pt) The following two questions ask you to complete client-side functions. Do not violate the abstraction barrier! Implement the `buy_stock` function

```
def buy_stock(stock_market, account, ticker):
    __________________________________________________________
    __________________________________________________________
```

```
def buy_stock(stock_market, account, ticker):


    __________________________________________________________

    __________________________________________________________
```

(f) (1.0 pt) Implement the `sell_stock` function

```
def sell_stock(stock_market, account, ticker):
    __________________________________________________________
    __________________________________________________________
```

```
def sell_stock(stock_market, account, ticker):

    __________________________________________________________

    __________________________________________________________
```

10. (7.0 points) Where's my peanut?

You are a squirrel on Berkeley campus, and one of your important daily jobs is to find yummy peanuts on open fields. *Definition*: a `field` in this context is a possibly nested list representing the field you are on. Its elements are either `'pebble'`, `'peanut'`, or another list representing a deeper field. You start from level 1. Every time you enter a nested list, level increases by 1. For example, the list `['pebble', ['pebble', ['pebble', 'peanut']], 'pebble']` has 3 levels, and the peanut is at level 3. **It's guaranteed that each `field` only contains 1 peanut.** Implement `find_peanut`, which takes in a list `field`, and a positive integer `limit`. It returns the level where the peanut lies if it does not exceed `limit`; otherwise, return `'No peanut found'`. *Hint*: `isinstance(x, list)` returns `True` if `x` is a list and `False` otherwise. For example:

```
>>> isinstance('peanut', list)
False
>>> isinstance(['peanut'], list)
True
```

```
def find_peanut(field, limit):
    """
    >>> field_1 = ['pebble', ['pebble', 'pebble', ['pebble', 'peanut']], 'pebble']
    >>> find_peanut(field_1, 3)
    3
    >>> find_peanut(field_1, 2)
    'No peanut found'
    >>> field_2 = ['pebble', ['pebble', ['pebble']], 'peanut']
    >>> find_peanut(field_2, 1)
    1
    >>> field_3 = ['pebble', ['pebble', ['pebble', ['pebble']], 'peanut'], 'pebble']
    >>> find_peanut(field_3, 3)
    2
    """
    def helper(curr_field, curr_level):
        if ____(a)____ or ____(b)____:
            return 'No peanut found'
        elif ____(c)____:
            return curr_level
        else:
            if isinstance(curr_field[0], list):
                result = helper(____(d)____, ____(e)____)
                if result != 'No peanut found':
                    return result
            return helper(____(f)____, ____(g)____)
    return helper(field, 1)
```

(a) (1.0 pt) Which of the following CANNOT fill in blank (a)?

○ `curr_field`

○ `not curr_field`

○ `curr_field == []`

○ `len(curr_field) == 0`

○ `bool(curr_field)`

(b) (1.0 pt) Assume that one of the valid options in the previous part is used to fill in blank (a). **Now, fill in blank (b).**

(c) (1.0 pt) Which of the following can fill in blank (c)? Choose all that apply.

□ `curr_field[0] == 'peanut'`

□ `curr_field[0] == ['peanut']`

□ `curr_field == 'peanut'`

□ `curr_field == ['peanut']`

□ `'peanut' in curr_field`

□ `['peanut'] in curr_field`

□ `len(curr_field) == 1`

(d) (1.0 pt) Fill in blank (d).

(e) (1.0 pt) Fill in blank (e).

(f) (1.0 pt) Which of the following can fill in blank (f)?

○ `curr_field`

○ `curr_field[0]`

○ `curr_field[1:]`

(g) (1.0 pt) Fill in blank (g).

11. (8.0 points) Seal of Approval :3

Your favorite animal are seals, and because you love them so much you want to implement a class to model and simulate them. You start off by writing a `Seal` class.

```
class Seal:
    qualities = ['round', 'cute', 'smiley']

    def __init__(self, name):
        self.name = name
        self.qualities = Seal.qualities[:] # copy of the Class attribute qualities

    def compliment(self, new_quality):
        """
        >>> seal_friend = Seal('fren')
        >>> seal_friend.qualities
        ['round', 'cute', 'smiley']
        >>> Seal.qualities
        ['round', 'cute', 'smiley']
        >>> seal_friend.compliment('fluffy')
        fren you are so round
        fren you are so cute
        fren you are so smiley
        fren you are so fluffy
        >>> seal_friend.qualities
        ['round', 'cute', 'smiley', 'fluffy']
        >>> Seal.qualities
        ['round', 'cute', 'smiley']
        """
        ________________________
        for quality in ___________________________:
            print( ________________ + " you are so " + quality)
```

(a) (3.0 pt) The constructor has been given to you; your job is to implement the method `compliment`, which takes in `new_quality`: a string representing a new positive quality about the seal.

`compliment` will add `new_quality` to the end of the `qualities` attribute of the given Seal instance. Then, each of the seal's qualities is printed out as a compliment according to the format in the doctest.

You should add `new_quality` such that subsequent calls to `compliment` remember previous compliments given to the Seal instance.

```
def compliment(self, new_quality):

    ____________________________________________________________

    for quality in ____________________________________________:

        print(______________________ + " you are so " + quality)
```

(b) **(3.0 pt)** Sometimes seals have caretakers who take care of many seals. In this part, we will be implementing a `CareTaker` class.

```
class CareTaker:
    def __init__(self, names, compliments):
        """
        >>> names = ['Tsubaki', 'Yuki']
        >>> compliments = ['smart', 'fluffy']
        >>> care_taker = CareTaker(names, compliments)
        >>> care_taker.seal_compliments
        {'Tsubaki': 'smart', 'Yuki': 'fluffy'}
        """
        d = ______________________
        for ______________________:
            -------------------------
        -------------------------

        def compliment(self, seal):
            """
            >>> yuki = Seal('Yuki')
            >>> whiskers = Seal('Whiskers')
            >>> care_taker = CareTaker(['Tsubaki', 'Yuki'], ['smart', 'fluffy'])
            >>> care_taker.compliment(yuki)
            Yuki you are so round
            Yuki you are so cute
            Yuki you are so smiley
            Yuki you are so fluffy
            """
            _________________.compliment(_________________)
```

Each `CareTaker` object has an instance attribute `seal_compliments`, which is a dictionary of the names of their seals mapped to a unique compliment for each seal. The constructor is given `names`, a list of seal names, and `compliments`, a list of compliments corresponding to each seal name (`compliments[0]` corresponds to `names[0]`, `compliments[1]` corresponds to `names[1]` . . .). In the constructor, create the instance attribute `seal_compliments` and populate it with a dictionary mapping seal name to its compliment.

```
def __init__(self, names, compliments):

    d = ________________________________________________________

    for _______________________________________________________:

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

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

(c) (2.0 pt) Implement `compliment` for the CareTaker class, which takes in a `Seal` object `seal`, and compliments the seal with their corresponding compliment from `self.compliments`. You can assume the `seal` will be one of the CareTaker's seals.

```
def compliment(self, seal):
    """
    >>> yuki = Seal('Yuki')
    >>> whiskers = Seal('Whiskers')
    >>> care_taker = CareTaker(['Tsubaki', 'Yuki'], ['smart', 'fluffy'])
    >>> care_taker.compliment(yuki)
    Yuki you are so round
    Yuki you are so cute
    Yuki you are so smiley
    Yuki you are so fluffy
    """
    _________________.compliment(_________________)
```

```
def compliment(self, seal):

    _____________________.compliment(__________________________)
```

No more questions.