CS 88 Fall 2021 Computational Structures in Data Science

MIDTERM

INSTRUCTIONS

- You have 120 minutes to complete the exam. **Do NOT open the exam until you are instructed to do so!**
- You **must not** collaborate with anyone inside or outside of CS88.
- 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 CS88 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.
- You **may not** use example functions defined on your study guide unless a problem clearly states you can.
- For fill-in-the-blank coding problems, we will only grade work written in the provided blanks. You may only write one Python statement per blank line, and it must be indented to the level that the blank is indented.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.
    - **Online Exams: You may start you exam as soon as you are given the password.**
    - **You may have a digitial version of the CS88 Reference Sheet, or the PDF, but no other files.**
    - Open Reference Sheet

1. (5.0 points) WWPD

For each of the expressions in the table 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 a function is outputted, write "function". Your answers must fit within the boxes provided. Work outside the boxes will not be graded.

(a) (1.0 pt)

```
>>> False or 1 and 8 or True
```

(b) (2.0 pt)

```
>>> lucky = (lambda x: lambda y: (y - x) % 8 == 0)(8)

>>> def mystery(x, y):
...     if lucky(x):
...         print('ready')
...     while x > y:
...         x = x // 10
...         print(x)
...     return 'go'

>>> mystery(40, 2)
```

(c) (2.0 pt) Note the variable `lucky` and the `mystery` function are redefined here for your convenience.

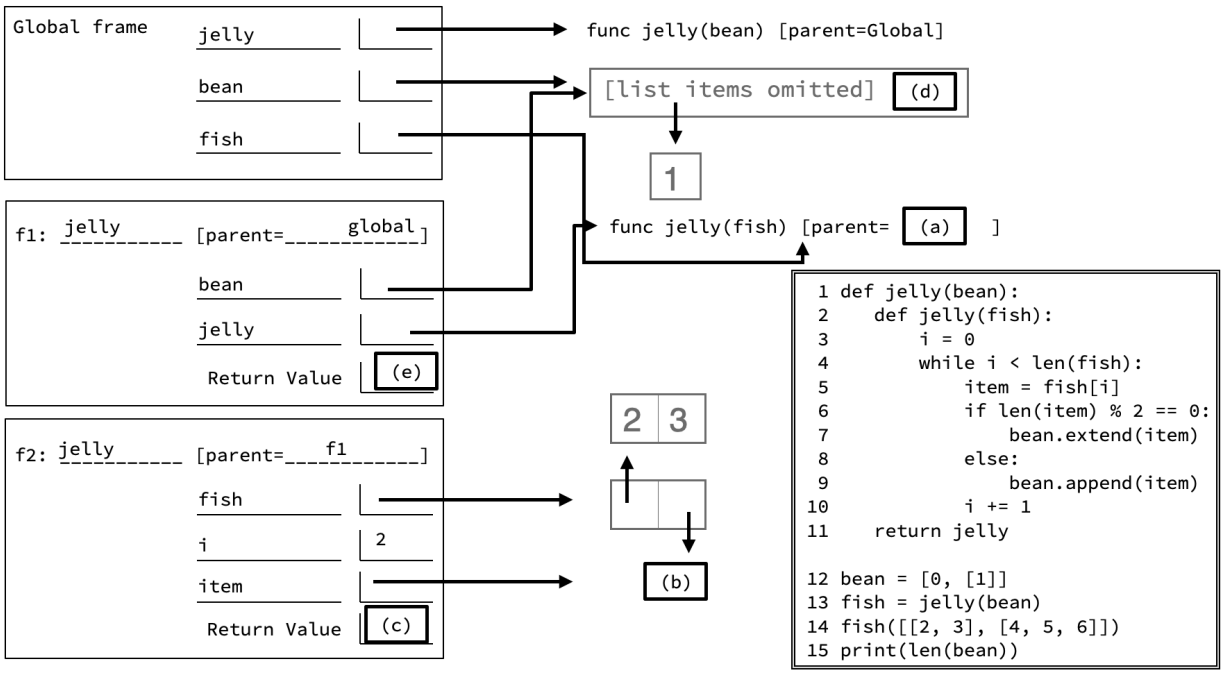
```
>>> lucky = (lambda x: lambda y: (y - x) % 8 == 0)(8)

>>> def mystery(x, y):
...     if lucky(x):
...         print('ready')
...     while x > y:
...         x = x // 10
...         print(x)
...     return 'go'

>>> print('a', mystery(7, 9))
```

2. (7.0 points) Environment Diagram Analysis

Fill in the blanks to complete the environment diagram. All the code used is in the box to the right. *Some arrows have been removed from the diagram.* You may wish to draw in the arrows, but it is not required.



Question 2 Environment Diagram

(a) (1.0 pt) **(a)** What is the parent frame of the `jelly` function on line 2?

(b) (1.0 pt) **(b)** What is the final value of `item` in the `f2` frame when the environment diagram is complete?

(c) (1.0 pt) **(c)** What is the return value of the `f2` frame?

(d) (2.0 pt) **(d)** For the variable `bean` in the `f1` frame, what is `bean[4]` when the environment diagram is complete?

(e) (1.0 pt) (e) What is the return value of frame `f1`?

(f) (1.0 pt) Line 15 What is `len(bean)` in the global frame?

3. (5.0 points) Just Keep Summing

Implement the function `sum_until`, which takes in an integer `total`, and returns a one-argument function, `add_num`. The `add_num` function keeps accepting integers until the sum of all the integers it has accepted reaches or exceeds the total, in which case it returns `True` or `False` respectively.

The expected behavior of the function is detailed in the doctests below.

```
def sum_until(total):
    """
    >>> f = sum_until(0)
    >>> f(0) # 0 = 0
    True
    >>> f(-7) # -7 < 0, so continue to accept more numbers
    <function sum_until.<locals>.add_num at ....>
    >>> f(-7)(5)(2) # -7 + 5 + 2 = 0
    True
    >>> f(-7)(5)(8) # -7 + 5 + 8 = 6 > 0
    False
    >>> g = sum_until(-5)
    >>> g(-6)(3) # -6 + 3 = -3 > -5
    False
    >>> g(-11)(-2)(-2)(4)(6) # -11 + -2 + -2 + 4 + 6 = -5
    True
    """
    def add_num(x):
        if x > total:
              return ______________Part A_________________
        elif ______________Part B_________________:
            return ______________Part C_________________
        return sum_until(______________Part D_________________)
    return add_num
```

(a) (1.0 pt) Fill in the code for Part A.

(b) (1.0 pt) Fill in the code for Part B.

(c) (1.0 pt) Fill in the code for Part C.

(d) (2.0 pt) Fill in the code for Part D.

4. (7.0 points) Flip Flop

Implement the `flip_flop` function which takes in a non negative number `n` and computes the value generated when alternating between adding then subtracting each of the digits in `n` from left to right.

You may use the `get_length` function that takes in an integer `n` and returns the number of digits in `n`. This function's implementation is hidden, but you can assume it works correctly.

```
def get_length(n):
    """
    Helper function that computes the number of digits in an
    integer.
    >>> get_length(7)
    1
    >>> get_length(123)
    3
    >>> get_length(454545)
    6
    """
    # Implementation hidden

def flip_flop(n):
    """
    >>> flip_flop(124) # 1 + 2 - 4
    -1
    >>> flip_flop(7315) # 7 + 3 - 1 + 5
    14
    >>> flip_flop(61323) # 6 + 1 - 3 + 2 - 3
    3
    """
    if ____________________________________:
        ____________________________________
    else:
        last_digit = ____________________________________
        if ____________________________________:
            return ____________________________________
        else:
            return ____________________________________
```

(a) (7.0 pt) Write the fully *completed* `flip_flop` function below using the skeleton code provided. You may not add, change, or delete lines from the skeleton code.

Don't forget to use the helper function `get_length(n)`.

```
def flip_flop(n):

    if ____________________________________________________:

        ________________________________________

    else:

        last_digit = ____________________________________________________

        if ____________________________________________________:

            return ____________________________________________________

        else:

            return ____________________________________________________
```

5. (8.0 points) Cra88y Crawl

You are working at an amusement park this summer and are in charge of the Cra88y Crawl ride! Complete the following questions to collect information that can improve visitors' experience on this ride!

The line of visitors is represented as a list of two element tuples.

- The first element in the tuple is the time when the visitor joined the line.
- The second element is the amount of time the visitor is expected to wait to begin their ride. Time is represented as the number of minutes elapsed since the amusement park opened for that day.

For example, the tuple `(80, 40)` represents that a visitor joined at the 80th minute and is expected to wait in line for 40 minutes.

(a) (3.0 pt) Complete the return statement for the `expected_start_times` function that, given a line, returns a list of the times (represented in minutes) at which every visitor is expected to start their ride!

```
def expected_start_times(line):
    """
    >>> expected_start_times([(80, 40), (105, 20)])
    [120, 125]
    >>> expected_start_times([(5, 15), (8, 12), (14, 6), (222, 3)])
    [20, 20, 20, 225]
    >>> expected_start_times([(4, 6), (5, 5), (5, 5), (6, 15), (100,
    20), (150, 0)])
    [10, 10, 10, 21, 120, 150]
    """
    return _____________________________________________
```

(b) (5.0 pt) Implement the `remove_frustrated_visitors` function that removes the tuples corresponding to visitors who are expected to wait more than `max_wait_time` minutes in line.

```
def remove_frustrated_visitors(line, max_wait_time):
    """
    >>> line_a = [(80, 40), (105, 20)] # format per tuple: (join time, expected wait time)
    >>> remove_frustrated_visitors(line_a, 15)
    >>> line_a
    []
    >>> line_b = [(5, 15), (8, 12), (14, 6), (222, 3)]
    >>> remove_frustrated_visitors(line_b, 10)
    >>> line_b
    [(14, 6), (222, 3)]
    >>> line_c = [(4, 6), (5, 5), (5, 5), (6, 15), (100, 20), (150, 0)]
    >>> remove_frustrated_visitors(line_c, 5)
    >>> line_c
    [(5, 5), (5, 5), (150, 0)]
    """
    position = 0
    while position < len(line):
        if ____________________________________:
            ____________________________________
        else:
            ____________________________________
```

Fill in the solution in the spce provided. You should not need to add or remove lines.

```
def remove_frustrated_visitors(line, max_wait_time):

    position = 0

    while position < len(line):

        if ____________________________________________________:

            ____________________________________________________

        else:

            ____________________________________________________
```

6. (7.0 points) Chef's Assistant

You are a chef managing orders for your restaurant! Every order consists of a food item and its quantity.

Implement the function `add_new_orders` which takes in a dictionary `all_orders` representing the orders a chef is currently assigned, and mutates it to include new orders from the list `new_orders`. Each element in `new_orders` is a tuple, where the first element is the food item and the second element is that item's quantity.

```
def add_new_orders(all_orders, new_orders):
    """
    >>> order = {'fries': 3, 'burger': 4}
    >>> add_new_orders(order, [('fries', 4), ('milk', 2)])
    >>> order == {'fries': 7, 'burger': 4, 'milk': 2}
    True
    >>> add_new_orders(order, [('fries', 2), ('taco', 1)])
    >>> order == {'fries': 9, 'burger': 4, 'milk': 2, 'taco': 1}
    True
    """
    if ________________________________________:
        return
    food_item = new_orders[0][0]
    quantity = new_orders[0][1]
    if ________________________________________:
        ________________________________________
    else:
        ________________________________________
    add_new_orders(__________________________, __________________________)
```

(a) (7.0 pt) Complete the skeleton code. You may not add, change, or delete lines from the skeleton code.

```
def add_new_orders(all_orders, new_orders):
    if ______________________________:
        return
    food_item = new_orders[0][0]
    quantity = new_orders[0][1]
    if ______________________________:
        ______________________________
    else:
        ______________________________
    add_new_orders(_________________, _________________)
```

7. (6.0 points) Smooooooooth

Your friend is trying to implement a function `smooth` which takes in a list of integers, `lst`, and returns a ***smooth*** version of the list, where smoothing a list means connecting each of the adjacent integers in the old list with consecutive integers.

For example, the smoothed version of `[1, 5, 3]` is `[1, 2, 3, 4, 5, 4, 3]`, and the smoothed version of `[1, 3, 4, 6]` is `[1, 2, 3, 4, 5, 6]`. There is guaranteed to be at least one number in the list and there will not be identical numbers consecutively, so `[1, 1, 1, 1]` is an invalid input. Your friend's code has 3 bugs which you need to find!

```
1. def smooth(lst):
2.     new_lst = []
3.     for i in range(len(lst) - 1):
4.         curr = lst[i]
5.         next = lst[i + 1]
6.         while curr != next:
7.             new_lst + [curr]
8.             if curr > next:
9.                 curr += 1
10.             else:
11.                 curr -= 1
12.     return new_lst
```

In each box: Identify *one* of the 3 unique bugs and explain how to fix each bug. You must specify the line number you would change or delete, or between which lines you would add a new line of code.

(a) (2.0 pt)

(b) (2.0 pt)

(c) (2.0 pt)

8. (10.0 points) Overlap

You are working on representing users on a new messaging app called Overlap, which focuses on users' interests as a way to connect them! Every `User` has a name, a list of interests, and a list of followers!

Implement the following functions of the `User` class based on the descriptions below. For a given `User`, `a_user`, we should be able to execute:

- `a_user.add_follower(other_user)`: Takes in another User object `other_user` and adds it to this user's followers list if they are not already in the list
- `a_user.mutual_interests(other_user)`: Takes in another User object `other_user` and returns a list containing all of this user's interests that are shared with `other_user`
- `a_user.find_new_interest()`: Returns a string representing a new interest for this User. To determine this new interest, first identify this user's follower that has the largest number of mutual interests with this user. Then return a randomly selected interest from this follower. But be careful, this randomly selected interest must not already exist in this user's interests (otherwise it would not be new!). Assume that the user's interests and followers are non-empty.

```
import random

class User:
    def __init__(self, name, interests=[]):
        self.name = name
        self.interests = interests
        self.followers = []

    def add_follower(self, other_user):
        """ Part A """

    def mutual_interests(self, other_user):
        """ Part B """

    def separate_interests(self, other_user):
        """ Implementation not shown. Assume that this function takes in
        another User object and returns a list containing all
        of this user's interests that are NOT shared with other_user"""

    def find_new_interest(self):
        """ Part C """
```

(a) **i.** **(3.0 pt)** Implement the `add_follower` method which takes in another `User` object, `other_user`, and adds that user to this user's followers list if they are not already in the list.

```
def add_follower(self, other_user):
    """
    >>> u1 = User('bob', ['cooking', 'archery', 'tv'])
    >>> u2 = User('alice', ['shopping', 'guitar', 'cooking'])
    >>> u3 = User('mike', ['poker', 'tv', 'cooking'])
    >>> u1.add_follower(u2)
    >>> u1.add_follower(u3)
    >>> u1.add_follower(u2) # no change
    >>> [u.name for u in u1.followers]
    ['alice', 'mike']
    """
    if ______________________________________________:
        ______________________________________________
```

Write the fully `add_follower` function below using the skeleton code provided. You may not add, change, or delete lines from the skeleton.

```
def add_follower(self, other_user):

    if ______________________________________________:

        ______________________________________________
```

(b) **i.** **(3.0 pt)** Implement the `mutual_interests` method which takes in another `User` object, `other_user`, and returns a list containing all of this user's interests that are shared with `other_user`. (Again, there is *no need* to copy the doctests.)

```
def mutual_interests(self, other_user):
    """
    >>> u1 = User('bob', ['cooking', 'archery', 'tv'])
    >>> u2 = User('alice', ['shopping', 'guitar', 'cooking'])
    >>> u3 = User('mike', ['poker', 'tv', 'cooking'])
    >>> u1.mutual_interests(u2)
    ['cooking']
    >>> u1.mutual_interests(u3)
    ['cooking', 'tv']
    """
    return _____________________________________________
```

Complete the `return` statement of the `mutual_interests` function below.

(c) i. (4.0 pt) Implement the `find_new_interest` method that returns a string representing a new potential interest for this `User`. To determine this new interest, first identify this user's most similar follower that has the largest number of mutual interests with this user. Then return a randomly selected interest from this follower. But be careful, this randomly selected interest must not already exist in this user's interests (otherwise it would not be new!).

For this problem, assume that the user's interests and followers are non-empty. Note that the `separate_interests` function (see `User` class skeleton) may be helpful here. You may use `random.choice(lst)` to return a radomly selected item from a list, `lst`.

```
def find_new_interest(self):
    """
    >>> u1 = User('bob', ['cooking', 'archery', 'tv'])
    >>> u2 = User('alice', ['shopping', 'guitar', 'cooking']) # has one in common with bob
    >>> u3 = User('mike', ['poker', 'tv', 'cooking']) # has two in common with bob
    >>> u1.add_follower(u2)
    >>> u1.add_follower(u3)
    >>> u1.find_new_interest()
    'poker'
    """
    most_similar_follower = max(
        ____________________________________,
        key = ________________________________________
    )
    return random.choice(_____________________________________________)
```

Write the fully *completed* `find_new_interest` function below using the skeleton code provided. You may not add, change, or delete lines from the skeleton code.

```
def find_new_interest(self):

    most_similar_follower = max(

        ____________________________________,

        key = ________________________________________

    )

    return random.choice(_____________________________________________)
```

No more questions.