CS 88
Spring 2020

Computational Structures in Data Science

Midterm Solutions

INSTRUCTIONS

- The exam is worth 60 points, across 6 questions.
- You should have 5 sheets (9 sides) in your exam booklet.
- You have 2 hours to complete the exam. **Do NOT open the exam until you are instructed to do so!**
- The exam is closed book, closed notes, closed computer, closed calculator, except **one** hand-written 8.5" × 11" crib sheet of your own creation and the official CS 88 Midterm study guide.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Full Name	
Student ID Number	
Official Berkeley Email (@berkeley.edu)	
TA	
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 use built-in Python functions that do not require import, such as `min`, `max`, `pow`, `len`, and `abs`.
- 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.

1. **(10 points) What Would Python Display?**

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", but include all output displayed before the error. If evaluation would run forever, write "Forever". To display a function value, write "Function". The first two rows have been provided as examples.

The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

Assume that you have first started `python3` and executed the statements on the left.

```
x = 8
y = 88
z = 888
s = 'cs'
f = ( lambda x: lambda y:
        lambda z: x + y(z) )(2)

def strange(x):
    print('1/4')
    print(x + x, print(5))
    return
    print(1/0)

buffer = 3
def foo(x):
    def bar(g, t):
        print("mustard")
        if x % 3 == 0:
            print(g(x))
        else :
            print(t(x))
    print("ketchup")
    return bar
```

Expression	**Interactive Output**

`g = print('hello world')`
`print(g)`
	hello world
	None

`s + y`	Error
`f(lambda x: x * x)(2)`	`6`

`strange(7)`	`1/4`
	`5`
	`14 None`

`a = foo(5)`	`ketchup`

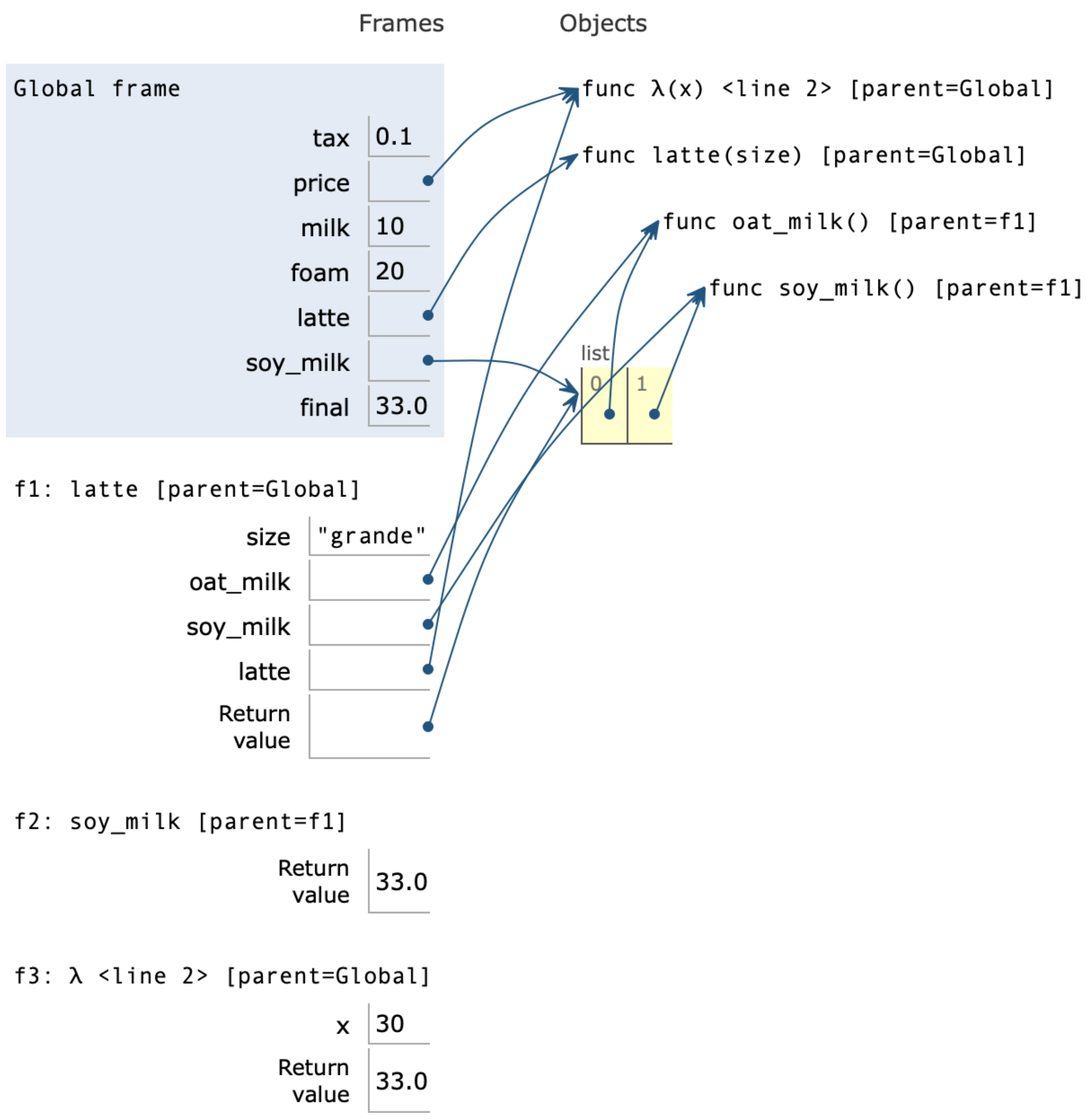
`b = a(lambda x: x + buffer,`
`lambda x: (x * x) + buffer)`
	`mustard`
	`28`

2. (10 points) A Brew-tiful Environment!

Fill in the environment diagram that results from executing the code on the right until the entire program is finished, an error occurs, or all frames are filled. You
A complete answer will:

- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.
- Draw any arrows as necessary.

```
 1  tax = 0.1
 2  price = lambda x: x * (1 + tax)
 3  milk = 10
 4  foam = 20
 5
 6  def latte(size):
 7      def oat_milk():
 8          return lambda: size
 9      def soy_milk():
10          return latte(milk + foam)
11      latte = price
12      return [oat_milk, soy_milk]
13
14  soy_milk = latte("grande")
15  final = soy_milk[1]()
```



3. (10 points) Counting?! It's All Greek to Me.

(3.1) Fill in the body of `find_digits` which takes a number n and a function $func$. $func$ takes in a single parameter and returns a Boolean. `find_digits` should return a new number that is built from all the digits that the function evaluates to `True`. The new number should keep all the digits in the same order that they were in n. If there are no digits that satisfy the function, return `0`.

In order to fill out the body of `find_digits`, we have provided the lines of code for you. However, these lines are not in the correct order! You should reorder them, and fill in any blank values. You may assume each line should be used once, and there is no need for additional code.

```
return_number = 0
return_number += current_digit * 10 ** ten_exponent
current_digit = n % 10
return return_number
if func(current_digit):
while (n > 0):
n = n // 10
ten_exponent += 1
ten_exponent = 0
```

```
def find_digits(n, func):
    """
    >>> find_digits(123456789, lambda x: x < 4)
    123
    >>> find_digits(1936382, lambda x: x > 5)
    968
    >>> find_digits(1111111, lambda x: x > 2)
    0
    """
    return_number = 0
    ten_exponent = 0
    while (n > 0):
        current_digit = n % 10
        n = n // 10
        if func(current_digit):
            return_number += current_digit * 10 ** ten_exponent
            ten_exponent += 1
    return return_number
```

(3.2) Now, write a higher order function divisible that when passed into `find_digits` will return the digits divisible by y. You may assume that `find_digits` is implemented correctly.

We have provided more lines than you may need to solve this function.

```
def divisible(y):
    """
        >>> divisible_by_three = divisible(3)
        >>> find_digits(987654321, divisible_by_three)
    963
        >>> divisible_by_ten = divisible(10)
        >>> find_digits(987654321, divisible_by_ten)
    0
        """
        return lambda x: x % y == 0
```

4. (10 points) I'm Just a Student, Sitting in Front of a Computer, Asking it to Solve My Game

We've gotten bored of playing Tic-Tac-Toe with our friends, so we've decided it's time to write a program to ensure we always win! So far, we've built some handy functions, and now it's your turn to complete our program.

Given the functions below, complete the function named `determine_win`, which returns whether or not the given player won the Tic-Tac-Toe game. In the Tic-Tac-Toe game, a player can win if their marker, $'X'$ or $'O'$ makes a straight line horizontally, vertically, or diagonally.

We have provided three functions, which you can assume are correct: `all_four_equal`, `column`, and `diagonal`. For full credit, your solution must make use of each of these functions. (Part of this question is to spend time reading and understanding these functions before moving on to your own solution. As long as you use these three, there's many valid solutions.)

```
def all_four_equal(pieces, player):
    """
    Returns true if the set of 3 pieces is the same as the player.
    >>> all_four_equal(['X', 'X', 'X'], 'O')
    False
    """
    return pieces[0] == pieces[1] == pieces[2] == player

def column(board, index):
    """
    Return the 0th, 1st, or 2nd column in a board.
    >>> column([['X', 'O', 'X'],
                ['O', 'X', 'X'],
                ['O', 'X', 'O']], 1)
    ['O', 'X', 'X']
    """
    return [ row[index] for row in board ]

def diagonal(board, index):
    """
    Return either 0th or 1st diagonal in a board.
    >>> diagonal([['X', 'O', 'X'],
                  ['O', 'X', 'X'],
                  ['O', 'X', 'O']], 1)
    ['X', 'X', 'O']
    """
    if index == 0:
        return [ board[0][0], board[1][1], board[2][2] ]
    else:
        return [ board[0][2], board[1][1], board[2][0] ]
```

```
def determine_win(board, player):
    """
    >>> determine_win([['X', 'O', 'X'],
                       ['O', 'X', 'X'],
                       ['O', 'X', 'O']], 'O')
    False
    >>> determine_win([['O', 'O', 'O'],
                       ['X', 'O', 'X'],
                       ['X', 'O', 'X']], 'O')
    True
    """
    for row in board:
        if all_four_equal(row, player):
            return True
    for col in range(3):
        if all_four_equal(column(board, col), player):
            return True
    return all_four_equal(diagonal(board, 0), player) or
        all_four_equal(diagonal(board, 1), player)
```

5. **(10 points) Atey Ate Already**

It's a lot more fun to think about food than take midterms, so let's look at the cheapest places to fulfill an order. Given the function `total_cost` and assuming it works as described, fill out `find_restaurant` to find the cheapest restaurant to fulfill the order.

Remember: Pay close attention to the doctests to guide your solution.

```
def totalCost(restaurant, order):
        """
        Function that returns the total cost of an order at a certain restaurant. Retu
      >>> totalCost('chipotle', [    burrito    ,     taco    ])
        11.96
        >>> totalCost(    sliver    , [    boba    ])
        -1.0
        """
        # We have omitted how this function works.
        return something_smart

def findRestaurant(restaurants, order):
    """
    Function that returns the cheapest restaurant and price as the first
    element of a list followed by the prices for each of the restaurants.
    In the case that no restaurant can fulfill the order, the first
    element should be ['None found!', -1].
    Hint: Use totalCost! In the case that two restaurants have the same price,
    keep the first restaurant.
    >>> findRestaurant(['chipotle', 'la burrita'], ['burrito', 'taco'])
    [['la burrita', 9.78], [['chipotle', 11.96], ['la burrita',9.78]]
    >>> findRestaurant(['sliver','cheeseboard'], ['boba'])
    [[None found!, -1.0], ['sliver', -1.0]['cheeseboard', -1.0]]
    """
    placesList = [ [restaurant, totalCost(restaurant, order)]
                    for restaurant in restaurants ]
    minCost = -1.0
    cheapestPlace = "None found!"
    for place in range(placesList):
        if place[1] != -1.0 and (place[1] < minCost or minCost == -1.0):
            minCost = place[1]
            cheapestPlace = place[0]
    return [cheapestPlace, minCost] + placesList
```

6. (10 points) Rooms within Rooms within Rooms

You are a Data Scientist hired by UC Berkeley to find the largest room on campus. In order to schedule midterms, your job is return the room and its capacity. The data on all the rooms plus capacity is in a weird format of three element lists, where the first element is the room, the second element is the capacity, and the third element is either the rest of the data or None. Assume that the capacity of each of the rooms is unique.

That is, the data look like `['Room', Number, [...]]`.

Use the following lines of code to fill in the body of the function. You will need to fill in the blanks of the lines provided. Some lines are optional.

```
return [rooms[0], rooms[1]]
largest_left = find_largest(______________________)
if rooms[2] == ______________:
if largest_left[1] > rooms[1]:
return ____________________
return ____________________
else: # this line is optional, depending upon your solution
else: # this line is optional, depending upon your solution
```

```
def find_largest(rooms):
    """
    Return the largest room from a weirdly nested list.
    You can assume rooms is always 3 items long.
    >>> rooms = ['Evans', 150, ['Soda', 200, ['Wheeler', 700, ['Stadium', 50000, None]
    >>> find_largest(rooms)
    ['Stadium', 50000]
    >>> find_largest(['Evans', 150, ['Hearst Annex', 50, None]])
    ['Evans', 150]
    """
    if rooms[2] == None:
        return [rooms[0], rooms[1]]
    largest_left = find_largest(rooms[2])
    if largest_left[1] > rooms[1]:
        return largest_left
    return [rooms[0], rooms[1]]
```