Seminar Worksheet 15 Solution: Tuples

CS 101 Algorithmic Problem Solving

Fall 2023

Name(s): ____________________

HU ID *(e.g., xy01042)*: ____________________

1. Reorder the Names

A professor is given a list of student names where each student's name is represented as a tuple of 3 elements. But tuple contains the name in reverse order, meaning the last name is placed first then middle name then the first name. She wants to have a list of names in the correct order, having first name first and so on.

Write a function that takes list of tuples l as an input and returns the list with the correct order.

Constraints

- $len(l) \in \mathbb{Z}$
- $1 \leq len(l) \leq 200$

Interaction

The input comprises of a single line containing a list l.

The output must also contain a list, with new order of elements in the tuple.

Sample

Input	Output
[("Radcliffe","Jacob","Daniel"), ("Watson", "Charlotte","Emma")]	[("Daniel", "Jacob","Radcliffe"), ("Emma", "Charlotte", "Watson")]

In the sample case, the list returned contains each tuple with their order reversed.

Exercise

In the space provided, indicate the outputs for the given inputs.

Input	Output
[("Felton","Andrew","Thomas"), ("Wright", "Francesca","Bonnie")]	[("Thomas","Andrew","Felton"), ("Bonnie", "Francesca","Wright")]
[("Hiddleston", "William", "Tom")]	[("Tom", "William", "Hiddleston")]

Problem Identification

Briefly explain the underlying problem you identified in the above question that led you to your solution.

Answer: Function that takes names (as tuples) and returns a list of names that are reversed to get the desired order

Input: l
Output: Updated list that contains names in the form of tuples shifted, to get the first name first, then the middle name and finally the last name

Pseudocode

```
#Function Definition
def last_name_first(names):
    updated_names = []
    for name in names:
        name_tuple = (name[-1],) + name[:-1]
        updated_names.append(name_tuple)
    return updated_names

#Function Call
print(last_name_first(names))
```

Dry Run

Using any one of the inputs provided in the Exercise section above, dry run your pseudocode in the space below.

Input: $l = [(\text{“}Felton\text{”}, \text{“}Andrew\text{”}, \text{“}Thomas\text{”}), (\text{“}Wright\text{”}, \text{“}Francesca\text{”}, \text{“}Bonnie\text{”})]$

name	name_tuple	updated_names
(“Felton”,“Andrew”,“Thomas”)	(“Thomas”,“Andrew”,“Felton”)	[(“Thomas”,“Andrew”,“Felton”)]
(“Wright”,“Francesca”,“Bonnie”)	(“Bonnie”,“Francesca”,“Wright”)	[(“Thomas”,“Andrew”,“Felton”), (“Bonnie”,“Francesca”, “Wright”)]

Updated name list is printed after function returns it, which is the expected output

2. Surprise!

You step into a bustling airport terminal, where travelers are waiting for their flights. The seats are neatly arranged in rows and columns, with a distance of 1 m. After finding an available seat, you notice two of your friends waiting in the boarding area. Curious to catch up, you count the seats to determine their location.

As you figure out their row and chair number, you contemplate which friend is close enough for a surprise high-five without causing a scene.

Given the row and chair number of each friend as tuples $p_2 = (r, c), p_3 = (r, c)$ as well as your own row and chair number $p_1 = (r, c)$, figure out who to go to first.

Write a function called *distance* that takes three tuples as arguments p_1, p_2, p_3, and returns the number of the friend who you will go to first.

Hint: Use Distance Formula: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Constraints

- $0 \leq r, c \leq 15$

Interaction

The input comprises of a single line containing 3 space-separated tuples denoting the values of p_1, p_2 and p_3 respectively.

The output must contain a single number (either 2 or 3) denoting the friend you would say high to first. If they both are at an equal distance then print -1.

Sample

Input	Output
(0,2) (4,2) (0,0)	3
(4,0) (4,2) (2,0)	-1

In the first case, p_2 is 4 meters away while p_3 is 2 meters away, hence second friend, i.e. p_3 will be surprised first.

In the second case, p_2 is 2 meters away while p_3 is also 2 meters away, hence -1 will be the output.

Exercise

In the space provided, indicate the outputs for the given inputs.

Input	Output
(12,2) (6,2) (10,4)	3
(1,1) (1,7) (5,6)	2

Problem Identification

Briefly explain the underlying problem you identified in the above question that led you to your solution.

Answer: Function that takes positions of friends (as tuples) and returns the chosen friend

Input: p_1, p_2, p_3
Output: Using distance formula, calculate distance between p_1 and p_2, and between p_1 and p_3, and returns the one with smallest distance

Pseudocode

```
import math

#Function Definition
def distance(p1, p2, p3):
    d12 = math.sqrt(((p2[0] - p1[0])**2) + ((p2[1] - p1[1])**2))

    d13 = math.sqrt(((p3[0] - p1[0])**2) + ((p3[1] - p1[1])**2))

    if d12 == d13:
        return -1
    elif d12 < d13:
        return 2
    else:
        return 3

#Function Call
print(distance(p1, p2, p3))
```

Dry Run
Using any one of the inputs provided in the Exercise section above, dry run your pseudocode in the space below.

Input: $p_1 = (12, 2), p_2 = (6, 2), p_3 = (10, 4)$

d12	d13	d12 == d13?	d12 < d13?	d12 > d13?
6.0	2.8284271247461903	False	False	True

As d13 gives the smallest distance, hence function returns 3, which is printed in the main program, as expected

3. Let's Save Some Money!

Fatima decides to make a daily plan for saving up money for a week. She starts with certain amount which is represented as a tuple containing two values, x = (dollars, cents). She plans to save certain amount also represented as a tuple y = (dollar, cents) every day.

Write a function called $save_money$ that shows how her initial saving increases each day for a week by returning a list of 7 tuples. Function takes x and y as arguments.

Constraints

- $dollars, cents \in \mathbb{Z}$
- $0 \leq dollars, cents \leq 1000$

Interaction

The input comprises of a single line containing 2 space-separated tuples denoting x and y respectively.

The output must contain a single list containing tuples that show how her savings increase each day.

Sample

Input	Output
(2,30) (10,0)	[(12,30), (22,30), (32,30), (42,30), (52,30), (62,30), (72,30)]
(2,30) (0,0)	[(2,30), (2,30), (2,30), (2,30), (2,30), (2,30), (2,30)]

In the first case, $x = (2, 30)$. There are 2 dollars and 30 cents to start with. Each day she will be adding $y = (10, 0)$ to x, which is 10 dollars and 0 cents. After day 1, the updated savings will be dollars: 2 + 10 = 12, and cents: 30 + 0 = 30, so the first day's output tuple will be (12,30). Then next day, the updated savings will be 12 + 10 = 22 dollars, and 30 + 0 = 30 cents, so second tuple is (22,30). This way the third day's savings will be (32,30), fourth day will be (42,30), fifth day will be (52,30), sixth day will be (62,30) and final, i.e. seventh day, it will be (72,30) at the end of it all.

In the second case, $x = (2, 30)$. There are 2 dollars and 30 cents to start with. Each day she will be adding $y = (0, 0)$ to x, which is 0 dollars and 0 cents. After day 1, the updated savings will be dollars: 2 + 0 = 2, and cents: 30 + 0 = 30, so the first day's output tuple will be (2,30). As there are no daily savings, all the week's tuples will be (2,30).

Exercise

In the space provided, indicate the outputs for the given inputs.

Input	Output
(0,30) (10,0)	[(10,30), (20,30), (30,30), (40,30), (50,30), (60,30), (70,30)]
(10,10) (10,20)	[(20,30), (30,50), (40,70), (50,90), (60,110), (70,130), (80,150)]

Problem Identification
Briefly explain the underlying problem you identified in the above question that led you to your solution.

Answer: Function that takes the initial amount in dollars and cents (as a tuple) and the daily savings in dollars and cents (as a tuple), and returns the savings of the next 7 days

Input: x, y
Output: For each day, add the daily savings with the latest saved amount to get the savings. Continue this for 7 days to get a list of tuples as an output

Pseudocode

```
#Function definition
def save_money(x,y):
    savings_lst = []
    savings = x
    for i in range(7):
        dollar = savings[0] + y[0]
        cents = savings[1] + y[1]
        savings = (dollar, cents)
        savings_lst.append(savings)
    return savings_lst

#Function call
print(save_money(x,y))
```

Dry Run
Using any two of the inputs provided in the Exercise section above, dry run your pseudocode in the space below.

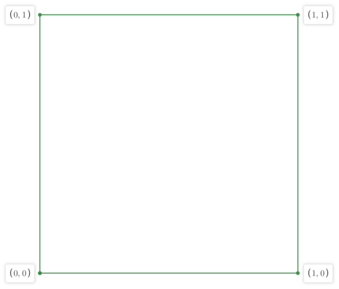
Input: $x = (10, 10), y = (10, 20)$

i	dollar	cents	savings	savings_lst
-	-	-	(10,10)	[ ]
0	20	30	(20,30)	[(20,30)]
1	30	50	(30,50)	[(20,30), (30,50)]
2	40	70	(40,70)	[(20,30), (30,50), (40,70)]
3	50	90	(50,90)	[(20,30), (30,50), (40,70), (50,90)]
4	60	110	(60,110)	[(20,30), (30,50), (40,70), (50,90), (60,110)]
5	70	130	(70,130)	[(20,30), (30,50), (40,70), (50,90), (60,110), (70,130)]
6	80	150	(80,150)	[(20,30), (30,50), (40,70), (50,90), (60,110), (70,130), (80,150)]

The function returns the list with seven days savings and main program prints it, which is the expected output

4. Line Cloud

A line cloud is a set of lines. Each line is defined by its two endpoints. The following figure shows a line cloud comprising four lines.



The lines in this line cloud have common endpoints but that need not always be the case.

We are going to represent a point as tuple of its coordinates, and a line as a tuple of its endpoints. A line cloud is then a list of lines. The above line cloud can be represented as $lines = [((0,0),(1,0)),((1,0),(1,1)),((1,1),(0,1)),((0,0),(0,1))]$. The point (0, 0) is connected to 2 different points: (0, 1) and (1, 0). It is said to have 2 neighbors.

Write a function called lineCloud that takes as argument a line cloud as described above. It returns a dictionary whose keys are the points in the line cloud. The value for each key is a list of the point's neighbors.

Constraints

- $len(lines) \in \mathbb{Z}$
- $0 \leq len(lines) \leq 100$

Interaction

The input comprises of a single line containing a list representing lines, where each line is tuple of points.

The output must contain a dictionary, with each unique point, i.e. tuple, as the key; and the list of neighboring points, as the value in the dictionary.

Sample

Input	Output
[((0,0), (1, 0)), ((1,0), (1, 1)), ((1,1), (0, 1)), ((0 ,1), (0, 0))]	{(0, 0): [(1, 0), (0, 1)], (1, 0): [(0, 0), (1, 1)], (1, 1): [(1, 0), (0, 1)], (0, 1): [(1, 1), (0, 0)]}

In the sample case, there are 4 distinct points hence, dictionary has 4 keys. (0,0) appears twice so it is connected to 2 different points, same is the case for the rest of points. We check where each point appears and then update the list accordingly.

Exercise

In the space provided, indicate the outputs for the given inputs.

Input	Output
[((0,0), (1, 1)), ((2,2), (3, 3)), ((4,4), (5, 5))]	{(0, 0): [(1, 1)], (1, 1): [(0, 0)], (2, 2): [(3, 3)], (3, 3): [(2, 2)], (4, 4): [(5, 5)], (5, 5): [(4, 4)]}
[((0,0), (1, 2)), ((1,2), (2,3)), ((2,3), (3,4)), ((3,4), (0,0))]	{(0, 0): [(1, 2), (3, 4)], (1, 2): [(0, 0), (2, 3)], (2, 3): [(1, 2), (3, 4)], (3, 4): [(0, 0), (2, 3)]}

Problem Identification
Briefly explain the underlying problem you identified in the above question that led you to your solution.

Answer: Function that takes a list of lines, where each line is specified by its end points, and produces a dictionary of neighbors of those lines

Input: *lines*
Output: An end point of a line may be connected to another line, thus making it the neighbor. For every end point, calculate the neighbors, i.e. the common end points between any two lines, and specify them in the form of dictionary

Pseudocode

```
#Function definition
def line_cloud(lines):
    neighbors = {}
    for line in lines:
        neighbors[line[0]] = neighbors.get(line[0], []) + [line[1]]
        neighbors[line[1]] = neighbors.get(line[1], []) + [line[0]]
    return neighbors

#Function Call
print(line_cloud(lines))
```

Dry Run
Using any one of the inputs provided in the Exercise section above, dry run your pseudocode in the space below.

Input: $lines = [((0,0),(1,2)),((1,2),(2,3)),((2,3),(3,4)),((3,4),(0,0))]$

line	line[0]	neighbors[line[0]]	line[1]	neighbors[line[1]]	neighbors
-	-	{ }	-	{ }	{ }
((0,0), (1, 2))	(0,0)	{(0,0):[(1,2)]}	(1, 2)	{(1,2):[(0,0)]}	{(0,0):[(1,2)], (1,2):[(0,0)]}
((1,2), (2,3))	(1,2)	{(1,2):[(0,0), (2,3)]}	(2,3)	{(2,3):[(1,2)]}	{(0,0):[(1,2)],(1,2):[(0,0), (2,3)], (2,3):[(1,2)]}
((2,3), (3,4))	(2,3)	{(2,3):[(1,2), (3,4)]}	(3,4)	{(3,4):[(2,3)]}	{(0,0):[(1,2)],(1,2):[(0,0),(2,3)], (2,3):[(1,2),(3,4)],(3,4):[(2,3)]}
((3,4), (0,0))	(3,4)	{(3,4):[(2,3), (0,0)]}	(0,0)	{(0,0):[(1,2), (3,4)]}	{(0,0):[(1,2),(3,4)],(1,2):[(0,0), (2,3)],(2,3):[(1,2),(3,4)], (3,4):[(2,3),(0,0)]}

The function returns the dictionary of neighbors and the main program prints it, which is the expected output

LET'S LEARN TO DEBUG

5. Find the Similarity

Given two tuples of numbers, Ali wants to see what numbers are common in both to see how similar the two tuples are.

Write a function that takes two tuples $t1$ and $t2$, and returns a tuple that has all common elements in the two tuples provided.

Constraints

- $0 \leq len(t1) \leq 10$
- $0 \leq len(t2) \leq 10$

Interaction

The input comprises of a single line containing 2 space-separated tuples $t1$ and $t2$.

The output must contain a single tuple comprising of common elements.

Sample

Input	Output
(4,6,8) (1,3,5)	()
(4,6,8,10) (1,2,3,4,5,6,8)	(4,6,8)

In the first case, there are no common elements hence an empty tuple is returned.

In the second case, 3 elements are common in both which are 4,6 and 8.

Proposed Solution

```
#Function definition
def count_common(t1,t2):
  common_elements_list = []
  for number in t1:
      if number in t2:
          common_elements_list.append(number)
          t2.remove(number)
  return tuple(common_elements_list)

#Function Call
print(count_common(t1,t2))
```

Dry Run

Using the inputs provided in the Sample section above, dry run the proposed code solution below.

Input: $t1 = (4, 6, 8), t2 = (1, 3, 5)$

t1	number	Is number in t2?	t2	common_elements_list
(4,6,8)	-	-	(1,3,5)	[ ]
(4,6,8)	4	No	(1,3,5)	[ ]
(4,6,8)	6	No	(1,3,5)	[ ]
(4,6,8)	8	No	(1,3,5)	[ ]

The list is empty, as expected, because there is nothing common in the tuples. It is converted into a tuple and returned to the main program, where it prints an empty tuple.

The function executes correctly, and returns the expected output, but what is the problem then? Check the other input to confirm the correctness

Input: $t1 = (4, 6, 8, 10), t2 = (1, 2, 3, 4, 5, 6, 8)$

t1	number	Is number in t2?	t2	common_elements_list
(4,6,8,10)	-	-	(1,2,3,4,5,6,8)	[ ]
(4,6,8,10)	4	Yes	Gives error as there is no remove function for tuples	Program fails

The program is never completed, as there are logical errors in it, so it must be corrected

Error Identification
Briefly explain the errors you identified in the proposed code solution. Mention the line number and the errors in each line.

Answer:
Line number 6, we need to rewrite the code with proper functionality of removing something from the tuple. However, as tuples are immutable, hence first convert it in a mutable data structure, let's say a list, remove the element, and then convert it back in tuple, with the selected value removed.

Correct Solution
Rewrite the lines of code you mentioned above with their errors corrected.

```
#Function definition
def count_common(t1,t2):
  common_elements_list = []
  for number in t1:
      if number in t2:
          common_elements_list.append(number)
          temp = list(t2)
          temp.remove(number)
          t2 = tuple(temp)
  return tuple(common_elements_list)

#Function Call
print(count_common(t1,t2))
```

Let's check again to see if it is working now

Input: $t1 = (4, 6, 8, 10), t2 = (1, 2, 3, 4, 5, 6, 8)$

t1	number	Is number in t2?	t2	common_elements_list
(4,6,8,10)	-	-	(1,2,3,4,5,6,8)	[ ]
(4,6,8,10)	4	Yes	(1,2,3,5,6,8)	[4]
(4,6,8,10)	6	Yes	(1,2,3,5,8)	[4,6]
(4,6,8,10)	8	Yes	(1,2,3,5)	[4,6,8]
(4,6,8,10)	10	No	(1,2,3,5)	[4,6,8]

The list now contains the common elements, and it can be converted into a tuple and return it to the main program, where it will be correctly printed. Hence the error has been corrected.