You can view this report online at : https://www.hackerrank.com/x/tests/1507478/candidates/49048558/report

Full Name:	Breeha Qasim
Email:	bq08283@st.habib.edu.pk
Test Name:	**CS102 - Lab 1 - Spring 2023**
Taken On:	13 Jan 2023 10:37:55 PKT
Time Taken:	800 min/ 200 min
Work Experience:	< 1 years
Invited by:	Aisha
Skills Score:	
Tags Score:	CS101 100/100 Functions 100/100 Python 100/100 Tuples 100/100

100%
400/400

scored in **CS102 - Lab 1 - Spring 2023** in 800 min on 13 Jan 2023 10:37:55 PKT

Recruiter/Team Comments:

No Comments.

Plagiarism flagged

We have marked questions with suspected plagiarism below. Please review.

	Question Description	Time Taken	Score	Status
Q1	Antisocial distancing > Coding	8 min 8 sec	100/ 100	✓
Q2	Last name first > Coding	28 min 7 sec	100/ 100	✓
Q3	Dude, Where's My Robot? > Coding	23 min 21 sec	100/ 100	!
Q4	Frog Race > Coding	55 min 55 sec	100/ 100	✓

QUESTION 1

Correct Answer

Score 100

Antisocial distancing > Coding

Python | CS101 | Tuples | Functions

QUESTION DESCRIPTION

Campus has reopened, and you find yourself in a large hall with seats arranged in a grid pattern of rows and columns. As per the established SOPs, each seat is at a distance of one meter from its neighbors in the front, in the back, to the left and to the right.

You find an empty seat to occupy, and quickly scan the hall for familiar faces. You spot, at some distance, your chuddy-buddy. You count the seats to figure out her row and chair number. You wonder if she's close enough for you to fly a paper plane up her nose.

Function Description

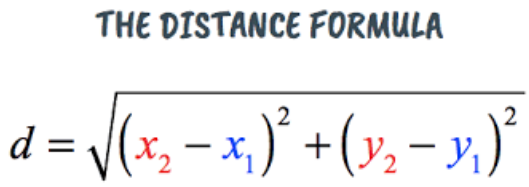
Complete the function *distance*. The function must calculate and return the distance between two positions given as (*row_number*, *chair_number*).

distance has the following parameter(s):

p1, p2: *tuple*s containing two *int*s

Hint

Distance *d* between two positions (x_1, y_1) and (x_2, y_2) is given by the following relationship:



Constraints

- None

▼ Input Format For Custom Testing

The first line contains a tuple with two ints, *p1*, denoting the first position as row and chair numbers.

The second line contains a tuple with two ints, *p2*, denoting the second position as row and chair numbers.

▼ Sample Case 0

Sample Input For Custom Testing

```
(3, 6)
(7, 6)
```

Sample Output

```
4.0
```

Explanation

Both you and your chuddy-buddy are in chair number 6. You're in row 3 while she's in row 7. She's four rows ahead of you, so you're 4.0 meters apart.

▼ Sample Case 1

Sample Input For Custom Testing

```
(4, 5)
(2, 9)
```

Sample Output

```
4.47213595499958
```

Explanation

You're seated in row 4, chair 5. Your chuddy-buddy is seated in row 2, chair 9. You're two rows and four chairs apart. Using the relationship above, you may calculate the distance to be about 4.47 meters.

INTERVIEWER GUIDELINES

```
def distance(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    d = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
```

```
    return d
```

CANDIDATE ANSWER

Language used: **Python 3**

```python
import math
def distance(p1, p2):
    d=0
    x2= in2[0]
    y2=in2[1]
    x1=in1[0]
    y1=in1[1]
    x_difference= x2-x1
    y_difference=y2-y1
    d+=math.sqrt((x_difference)**2 + (y_difference)**2)
    return d

in1 = eval (input())
in2 = eval(input())
print(distance(in1, in2))

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	Success	10	0.0296 sec	7.95 KB
Testcase 1	Easy	Sample case	Success	10	0.034 sec	7.92 KB
Testcase 2	Easy	Sample case	Success	10	0.0338 sec	7.9 KB
Testcase 3	Easy	Sample case	Success	10	0.0389 sec	7.96 KB
Testcase 4	Easy	Sample case	Success	10	0.1014 sec	7.93 KB
Testcase 5	Easy	Hidden case	Success	10	0.037 sec	7.91 KB
Testcase 6	Easy	Hidden case	Success	10	0.041 sec	7.96 KB
Testcase 7	Easy	Hidden case	Success	10	0.042 sec	7.73 KB
Testcase 8	Easy	Hidden case	Success	10	0.0522 sec	7.73 KB
Testcase 9	Easy	Hidden case	Success	10	0.0535 sec	7.84 KB

No Comments

QUESTION 2

Correct Answer

Score 100

Last name first › Coding

QUESTION DESCRIPTION

Challenge

Write a function named `last_name_first` that accepts a single parameter, `t`, which is passed a list of tuples. Each tuple contains a name in parts (first name, middle name, last name). Your function should modify each name so that the last name appears first in the tuple.

Note

This function modifies a list in place and, as such, should not return any useful value.

Sample interaction

```
>>> t = [('Ahmed', 'Dawood'), ('Haroon', 'Hussain', 'Fawad', 'Rasheed'),
('Muhammad', 'Faisal', 'Amin')]
>>> last_name_first(t)
>>> print(t)
[('Dawood', 'Ahmed'), ('Rasheed', 'Haroon', 'Hussain', 'Fawad'), ('Amin',
'Muhammad', 'Faisal')]
```

Input/Output

Input and output will be handled by HackerRank.

Constraints

`t` is a list of tuples, where each tuple has one or more strings in it.

INTERVIEWER GUIDELINES

```
def last_name_first(names):
    for p in range(len(names)):
        name = names[p]
        name = (name[-1], ) + name[:-1]
        names[p] = name
```

CANDIDATE ANSWER

Language used: **Python 3**

```
1  def last_name_first(t):
2      y=0
3      for x in t:
4          empty=() #lastname slicing
5          empty2=()
6          exclusion=x[len(x)-1]
7          empty+=(exclusion,)
8          #print(empty)
9          rest_names=x[:len(x)-1]
10         empty+=rest_names
11         t[y]=empty
12         y=y+1
13         #print(empty)
14
15
16
17
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	Success	10	0.0469 sec	7.7 KB
Testcase 1	Easy	Sample case	Success	10	0.0349 sec	7.7 KB
Testcase 2	Easy	Sample case	Success	10	0.0363 sec	7.77 KB
Testcase 3	Easy	Sample case	Success	10	0.0636 sec	7.89 KB
Testcase 4	Easy	Sample case	Success	10	0.0379 sec	7.88 KB
Testcase 5	Easy	Hidden case	Success	10	0.0508 sec	7.85 KB
Testcase 6	Easy	Hidden case	Success	10	0.05 sec	7.72 KB
Testcase 7	Easy	Hidden case	Success	10	0.0497 sec	7.75 KB

Testcase 8	Easy	Hidden case	Success	10	0.0292 sec	7.74 KB
Testcase 9	Easy	Hidden case	Success	10	0.0443 sec	7.69 KB

No Comments

QUESTION 3

Needs Review

Score 100

Dude, Where's My Robot? › Coding

QUESTION DESCRIPTION

Your neighbor's garden robot was hacked and moved from its charging pod where you had last left it. They cannot find the robot and need the grass trimmed right now for a party with their committee friends this evening. Luckily, they know of your skill with computers. You manage to retrieve the last set of instructions sent to the robot. Now you just have to figure out how far away the robot has moved.

The set contains *n* instructions. Each instruction is of the form *direction distance*. On receiving this instruction, the robot moves *distance* in *direction*. The robot executes the instructions one after the other.

Given the list of instructions, you want to find out the distance that that the robot has moved.

Input

The first line of the input contains *n (0 < n <100)*. The next *n* lines contain *direction* followed by *distance*, where direction is one of UP, DOWN, RIGHT, and LEFT. *distance* is an integer, *(-100 < distance < 100)*.

Output

Output the total distance that the robot has moved from its original distance once it has carried out the given instructions. Round the distance up to the nearest integer.

Examples

Input

```
4
UP 5
DOWN 3
LEFT 3
RIGHT 2
```

Output

```
3
```

Note

The robot moved 5 up, then 3 down, then 3 left, and then 2 right. Reluctantly, the robot is 2 up and 1 left from its original position. The distance is `math.sqrt(2*2 + 1*1)` which, rounded up, is 3.

Input

```
4
RIGHT 1
DOWN 0
LEFT -2
DOWN -4
```

Output

```
5
```

Note

The robot is 3 right and 4 up from its position. The distance is `math.sqrt(3*3 + 4*4)` which, rounded up, is 5.

Input

```
1
UP 0
```

Output

```
0
```

Note

The robot has not moved. The distance is 0.

INTERVIEWER GUIDELINES

Solution

```
import math
x = y = 0
for _ in range(int(input())):
    d, s = input().split()
    s = int(s)
    if d == 'UP':
        y += s
    elif d == 'DOWN':
        y -= s
    elif d == 'RIGHT':
        x += s
    elif d == 'LEFT':
        x -= s
print(math.ceil(math.sqrt(x*x + y*y)))
```

CANDIDATE ANSWER

Language used: **Python 3**

```
n=int(input())
import math
hori=0
y=0
for x in range(n):
    direction=input().split()
    dist=int(direction[-1])
    #print(b)

    if direction[0]=="UP":
        y+=dist
    elif direction[0]=="DOWN":
        y-=dist
    elif direction[0]=="LEFT":
        hori-=dist
    elif direction[0]=="RIGHT":
        hori+=dist
total_distance=math.sqrt((hori*hori)+(y*y))
final_output=math.ceil(total_distance)
print(final_output)
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	Success	10	0.0557 sec	7.91 KB
Testcase 1	Easy	Sample case	Success	10	0.0379 sec	8.12 KB
Testcase 2	Easy	Sample case	Success	10	0.0303 sec	8.03 KB
Testcase 3	Easy	Hidden case	Success	10	0.0345 sec	8.16 KB
Testcase 4	Easy	Hidden case	Success	10	0.0448 sec	8.06 KB
Testcase 5	Easy	Hidden case	Success	10	0.0239 sec	8.07 KB

Testcase 6	Easy	Hidden case	Success	10	0.0497 sec	7.9 KB
Testcase 7	Easy	Hidden case	Success	10	0.0339 sec	7.88 KB
Testcase 8	Easy	Hidden case	Success	10	0.0256 sec	8.1 KB
Testcase 9	Easy	Hidden case	Success	10	0.0523 sec	7.77 KB

No Comments

QUESTION 4

Correct Answer

Score 100

Frog Race › Coding

QUESTION DESCRIPTION

Two frogs named Frog Prime and Frogatron are in a well of depth 1000 meters. They decide to race each other to the top to see who gets out first. Each frog can jump up the wall of the well by a certain height but slips down a certain distance due to wetness on the wall. The frogs are not equally fit so the distance that they can jump up and end up sliding down is different from each other.

Frog Prime can jump `prime_up` meters at a time but slides down `prime_down` meters every time. Frogatron can jump `tron_up` meters at a time but slides down `tron_down` meters every time.

Both frogs take each jump at the same time and keep jumping till one of them clears the well and wins.

Your program should input the values, `prime_up`, `prime_down`, `tron_up`, and `tron_down` and then pass them to a function, `race`, which simulates the race. The function maintains a list of the positions of both frogs. Each position is stored as a *tuple* whose first element is the height of Frog Prime from the bottom of the well and second element is the height of Frogatron from the bottom of the well. As soon as a frog wins, the function prints the list, declares the winner, and announces the number of jumps the winner made.

Following is the output for some sample values.

Case 1

Input

`prime_up` = 50, `prime_down` = 2, `tron_up` = 1000, `tron_down` = 4

Output

[(0, 0), (48, 'OUT')]
Frogatron wins in 1 turns!

Explanation

The output list contains 2 positions. Both frogs begin at 0 so the first element is (0,0). Frog Prime jumps up 50 and slides down 2, thus gaining 48. At the same time, Frogatron jumps up 1000 and clears the well. As the well has been cleared, Frogatron does not slide down. The corresponding tuple in the list shows Frog Prime's position and that Frogatron has cleared the well. The winner was decided in 1 jump and no further jumps occurred so the list has no further elements.

Case 2

Input

`prime_up` = 520, `prime_down` = 100, `tron_up` = 410, `tron_down` = 20

Output

[(0, 0), (420, 390), (840, 780), ('OUT', 'OUT')]
Tie in 3 turns!

Explanation

The output list contains 4 positions. Both frogs begin at 0 so the first element is (0,0). Frog Prime jumps up 520 and slides down 100, thus gaining 420. At the same time, Frogatron jumps up 410 and slides down 20, thus gaining 390. The second element in the list is therefore (420, 390). At the next jump, each frog gains the same amount as in the previous jump. The next tuple in the list is thus (820, 780). At the next jump, each frog exceeds 1000 and gets out. The result is a tie decided in 3 jumps.

Input

4 space separated integer values. These are values of `prime_up`, `prime_down`, `tron_up`,

and `tron_down` respectively.

Constraints

- `prime_up`, `prime_down`, `tron_up`, and `tron_down` are of type *int*.
- `prime_up`, `prime_down`, `tron_up`, and `tron_down` are *> 0*.
- `prime_up` > `prime_down`
- `tron_up` > `tron_down`

INTERVIEWER GUIDELINES

Solution

```
def race(f1up, f1down, f2up, f2down):
    f1 = f2 = 0
    positions = [(0,0)]
    done = False
    while not done:
        f1 += f1up
        f2 += f2up
        jumps = len(positions)
        if f1 >= 1000:
            if f2 >= 1000:
                positions.append(('OUT', 'OUT'))
                winner = 'Tie in {} turns!'.format(jumps)
            else:
                f2 -= f2down
                positions.append(('OUT', f2))
                winner = 'Frog Prime wins in {} turns!'.format(jumps)
            print(positions)
            print(winner)
            done = True
        elif f2 >= 1000:
            f1 -= f1down
            positions.append((f1,'OUT'))
            winner = 'Frogatron wins in {} turns!'.format(jumps)
            print(positions)
            print(winner)
            done = True
        else:
            f1 -= f1down
            f2 -= f2down
            positions.append((f1,f2))

prime_up, prime_down, tron_up, tron_down = map(int, input().split())
race(prime_up, prime_down, tron_up, tron_down)
```

CANDIDATE ANSWER

Language used: **Python 3**

```
'''first_input=input().split()
prime_up=int(first_input[0])
prime_down=int(first_input[1])
tron_up=int(first_input[2])
tron_down=int(first_input[3])
l=[]
l.append((0,0))'''

def race(prime_up,prime_down,tron_up,tron_down,l):
    x=0
    y=0
    z=1
    x+=prime_up
```

```
    y+=tron_up
    count=1
    while z==1:
        if x>=1000 or y>=1000:
            break
        x-=prime_down
        y-=tron_down
        count+=1
        l.append((x,y))
        x+=prime_up
        y+=tron_up

    if x>=1000 and y>=1000:
       l.append(("OUT","OUT"))
       print(l)
       print("Tie in", str(count),"turns!")
    elif x>=1000:
        l.append(("OUT", y-tron_down))
        print(l)
        print("Frog Prime wins in", str(count), "turns!")
    elif y>=1000:
        l.append((x-prime_down,"OUT"))
        print(l)
        print("Frogatron wins in", str(count), "turns!")

first_input=input().split()
prime_up=int(first_input[0])
prime_down=int(first_input[1])
tron_up=int(first_input[2])
tron_down=int(first_input[3])
l=[]
l.append((0,0))
race(prime_up,prime_down,tron_up,tron_down,l)
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Medium	Sample case	Success	20	0.0362 sec	8.13 KB
Testcase 1	Medium	Sample case	Success	20	0.0417 sec	8.07 KB
Testcase 2	Medium	Sample case	Success	20	0.0348 sec	8.04 KB
Testcase 3	Medium	Sample case	Success	20	0.0591 sec	8.07 KB
Testcase 4	Medium	Sample case	Success	20	0.0763 sec	7.99 KB

No Comments