Recruiter Name: Kerri Hall
Email: kerri.hall@resourceemployment.com
Phone: 636.699.5381

Start a conversation like this:
Hi Kerri, I have found your contact from one of my friends, Do you have job openings for HireArt or Scale AI for a coding Specialist? I am looking for a job, and I have experience in.....

Assessment Question:

Question 1:
Two children, Jack and Mary, have devised an intriguing game. In this game, Jack sends a message to Mary. He ingeniously encodes the message in a manner that only Mary can decipher. He achieves this by reversing the letters within each word of the message. (However, each word stays in its original position). Additionally, he optionally adds one or more spaces at the message's beginning, end, and between each pair of words.

Put yourself in Mary's shoes. You have just received the encoded string and your mission is to decode it. Your task involves reversing each word, eliminating any unnecessary spaces at the beginning and end of the message, and ensuring that there is a maximum of only one space between any two words. Your final objective is to return the decoded string. Write the decode_message function that does this.

Example 1

Input: encoded_str = " olleH | epoh uoy era gniod llew"

Output: "Hello I hope you are doing well"

Example 2

Input: encoded_str = "ssendoog"

Question 2:

A group of n children is playing a game in which they are standing in a straight line. Each child has a hoop on the floor to stand in.
There are exactly as many hoops as children, hence n in total. On their shirts, each child has a number.

The game starts when the captain calls out a random number k.
Each child then moves to the next hoop on their right, wrapping around to the leftmost hoop when they reach the rightmost hoop.
This process is repeated k times.
Your task is to simulate this game by writing the function rotateArray. As input, you are given the number of children n, the value of k, and an array arr containing the numbers written on the children"s shirts in their original order from left to right. The function must return an array containing the new arrangement of the children after they have completed the moves.
Example 1
Input:

- n=5
- k=2
- arr = 12345

Output:
• 45123
Explanation:
• The value of k is 2, so each child moves 2 positions to the right in the line. The first child with number 1 moves to the position of number 3, the second child with number 2 moves to the position of number 4, and so on. The last two children with numbers 4 and 5 "go off" at the right end of the line and reappear at the left end in positions 1 and 2, respectively.
Example 2
Input:

- n=3
- k = 1
- Arr = 8 9 10

Output:

- 10 8 9

Interview Questions:

Question 1:

We are in a flower garden. The garden is represented as a collection of flame names.
We are trying to keep similar flowers close together, and we went to see what the farthest apart pair of the same flower are. Write a function that takes in a garden and returns the longest distance between any two flowers of the same type If no such distance exists, return -1.

Example:
Garden: I Ivy | Rue I Ivy | Yew | Rue | Dock | Iris | Rue I Lily I
Positions: 0 1 2 3 4 5 6 7 8

The longest distance is between Rue (position 1 and position 7), resulting in distance 6.
Test cases:
garden_1 = ["Ivy", "Rue", "Ivy", "Yew", "Rue", "Dock", "Iris", "Rue", "Lily"]
garden 2 = ["Rose"]
garden_3 = ["Ivy", "Rue", "Yew", "Arun", "Dock", "Iris", "Lily", "Reed", "Rose"]
garden 4 = ["Sage", "Rose", "Sage", "Reed", "Sage", "Lily", "Sage", "Iris", "Dock", "Sage", "Yew", "Rue",
', "Ivy"]
garden_5 = ["Dock", "Arum", "Yew", "Rue", "Ivy", "Ivy", "Rue", "Yew", "'Arum", "Dock", "Iris", "Lily", "Reed", "Rose", "Sage", "Dock"]
garden_6 = ["Rose", "Rose", "Rose", "Rose, "Rose", "Rose"]
garden_7 = ['Iris', "Iris"]
max_distance (garden_1) => 6
max_distance (garden 2) => -1
max distance (garden _3) => -1

```java
package FlowerMaximumDistance;

import java.util.HashMap;
import java.util.Map;

public class FlowerMaximumDistance {
   public static int maxDistance(String [] garden){
       Map<String,Integer> mapofGarden=new HashMap<>();
       int i=0;
       int maxDistance=-1;
       for(String s: garden){

           if(mapofGarden.containsKey(s)){
               int difference= i-mapofGarden.get(s);
               if(maxDistance<difference)
                   maxDistance=difference;
               i++;
               continue;
           }
           mapofGarden.put(s,i);
           i++;

       }
       return maxDistance;
   }


   public static void main(String[] args) {
```

```
        String[] garden1={"Ivy", "Rue" , "Ivy" ,"Yen"  ,"Rue","Dock", "Iras",
"Rue" , "Lay"};
        String[] garden2={"Rose"};
        System.out.println(maxDistance(garden1));
        System.out.println(maxDistance(garden2));
    }
}
```

Question 2:

You are going on a camping trip, but before you leave you need to buy groceries. To optimize your time spent in the store, instead of buying the items from your shopping list in order, you plan to buy everything you need from one department before moving to the next.

Given an unsorted list of products with their departments and a shopping list, return the time saved in terms of the number of department visits eliminated.

Example:
products = [
    ["Cheese",   "Dairy"],
    ["Carrots",   "Produce"],
    ["Potatoes",   "Produce"],
    ["Canned Tuna",   "Pantry"],
    ["Romaine Lettuce", "Produce"],
    ["Chocolate Milk", "Dairy"],
    ["Flour",   "Pantry"],
    ["Iceberg Lettuce", "Produce"],
    ["Coffee",   "Pantry"],
    ["Pasta",   "Pantry"],
    ["Milk",   "Dairy"],
    ["Blueberries",   "Produce"],
    ["Pasta Sauce",   "Pantry"]
]

list1 = ["Blueberries", "Milk", "Coffee", "Flour", "Cheese", "Carrots"]

For example, buying the items from list1 in order would take 5 department visits, whereas your method would lead to only visiting 3 departments, a difference of 2 departments.

Produce(Blueberries)->Dairy(Milk)->Pantry(Coffee/Flour)->Dairy(Cheese)->Produce(Carrots) = 5 department visits
New: Produce(Blueberries/Carrots)->Pantry(Coffee/Flour)->Dairy(Milk/Cheese) = 3 department visits

list2 = ["Blueberries", "Carrots", "Coffee", "Milk", "Flour", "Cheese"] => 2
list3 = ["Blueberries", "Carrots", "Romaine Lettuce", "Iceberg Lettuce"] => 0
list4 = ["Milk", "Flour", "Chocolate Milk", "Pasta Sauce"] => 2
list5 = ["Cheese", "Potatoes", "Blueberries", "Canned Tuna"] => 0

All Test Cases:
shopping(products, list1) => 2
shopping(products, list2) => 2
shopping(products, list3) => 0
shopping(products, list4) => 2
shopping(products, list5) => 0

Complexity Variable:
n: number of products

```java
package Shopping;

import java.util.*;

public class Shopping1 {
   public static int shopping(String[][] products, String[] shoppingList) {
       Map<String,String> productMap= new HashMap<>();
       for(String [] product: products){
           String item= product[0] ;
           String department=product[1];
           productMap.put(item,department);
       }

       int needTovisit=0;

       Set<String> visitedDepartment= new HashSet<>();
       String lastvisitedDepartment=null;

       for(String itemToPurchase: shoppingList){
           if(productMap.containsKey(itemToPurchase)){
               visitedDepartment.add(productMap.get(itemToPurchase));
           }

           if(!productMap.get(itemToPurchase).equals(lastvisitedDepartment)){
               needTovisit++;
           }
           lastvisitedDepartment=productMap.get(itemToPurchase);
       }
```

```
        return needTovisit-visitedDepartment.size() ;

    }


    public static void main(String[] args) {


        String[][] products = {
                {"Cheese", "Dairy"},
                {"Carrots", "Produce"},
                {"Potatoes", "Produce"},
                {"Canned Tuna", "Pantry"},
                {"Romaine Lettuce", "Produce"},
                {"Chocolate Milk", "Dairy"},
                {"Flour", "Pantry"},
                {"Iceberg Lettuce", "Produce"},
                {"Coffee", "Pantry"},
                {"Pasta", "Pantry"},
                {"Milk", "Dairy"},
                {"Blueberries", "Produce"},
                {"Pasta Sauce", "Pantry"}
        };

        String [] list1 = {"Blueberries", "Milk", "Coffee", "Flour", "Cheese",
"Carrots"};
        String[] list2 = {"Blueberries", "Carrots", "Coffee", "Milk", "Flour",
"Cheese"};
        String[] list3 = {"Blueberries", "Carrots", "Romaine Lettuce", "Iceberg
Lettuce"};
        String[] list4 = {"Milk", "Flour", "Chocolate Milk", "Pasta Sauce"};
        String[] list5 = {"Cheese", "Potatoes", "Blueberries", "Canned Tuna"};
        System.out.println(shopping(products,list1));
        System.out.println(shopping(products,list2));
        System.out.println(shopping(products,list3));
        System.out.println(shopping(products,list4));
        System.out.println(shopping(products,list5));
    }
}
```

Question 3:

We are working on a security system for a badged-access room in our company's building.

Given an ordered list of employees who used their badge to enter or exit the room, write a function that returns two collections:

All employees who didn't use their badge while exiting the room they recorded an enter without a matching exit (All employees are required to leave the room before the log ends.)

All employees who didn't use their badge while entering the room they recorded an exit withou a matching enter. (The room is empty when the Ing begins.)

9 18 Each collection should contain no duplicates, regardless of how many times a given employee matche the criteria for belonging to it.

```
records = {

    {"Paul", "enter"},
    {"Pauline", "exit"},
    {"Paul", "enter"},
    {"Paul", "exit"},
    {"Martha", "exit"},
    {"Joe", "enter"},
    {"Martha", "enter"},
    {"Steve", "enter"},
    {"Martha", "exit"},
    {"Jennifer", "enter"},
    {"Joe", "enter"},
    {"Curtis", "exit"},
    {"Curtis", "enter"},
    {"Joe", "exit"},
    {"Martha", "enter"},
    {"Martha", "exit"},
    {"Jennifer", "exit"},
    {"Joe", "enter"},
    {"Joe", "enter"},
    {"Martha", "exit"},
    {"Joe", "exit"},
    {"Joe", "exit"}

}
```

Expected Output: [[Paul, Steve, Curtis, Joe], [Pauline, Martha, Curtis, Joe]]

```
records1={
    {"Paul", "enter"},
```

{"Paul", "exit"}
}

Expected Output: [[], []]

records2={
{"Paul", "enter"},
{"Paul", "enter"},
{"Paul", "exit"},
{"Paul", "exit"}
}

Expected Output: [[Paul], [Paul]]

records3={
{"Raj", "enter"},
{"Paul", "enter"},
{"Paul", "exit"},
{"Paul", "exit"},
{"Paul", "enter"},
{"Raj","enter"}
}

Expected Output: [[Raj, Paul], [Paul]]

```
package EmployeeEnterExit;

import java.util.*;

public class EmployeeEnterExit {

   public static List<List<String>> listOfEmployees(String[][] records) {
       List<List<String>> result = new ArrayList<>();
       List<String> exitRecords = new ArrayList<>();
       List<String> entryRecords = new ArrayList<>();
       Set<String> recordChecker = new HashSet<>();
       for (String s[] : records) {
           String name = s[0];
           String status = s[1];

           if (status.equals("enter")) {
               if (recordChecker.contains(name)) {
                   entryRecords.add(name);
                   exitRecords.remove(name);
```

```
            }
            else {
                entryRecords.add(name);
                recordChecker.add(name);
            }
        } else if (status.equals("exit")) {
            if(recordChecker.contains(name)){
                entryRecords.remove(name);
                recordChecker.remove(name);
            }
            else{
                exitRecords.add(name);
            }

        }
    }
    result.add(removeDuplicates(entryRecords));
    result.add(removeDuplicates(exitRecords));
    return result;
}

public static List<String> removeDuplicates(List<String> duplicateStrings){
    List<String> nonDuplicateString= new ArrayList<>();
    for(String s: duplicateStrings){
        if(!nonDuplicateString.contains(s)){
            nonDuplicateString.add(s);
        }
    }
    return nonDuplicateString;
}

public static void main(String[] args) {
    String[][] records = {


            {"Paul", "enter"},
            {"Pauline", "exit"},
            {"Paul", "enter"},
            {"Paul", "exit"},
            {"Martha", "exit"},
            {"Joe", "enter"},
            {"Martha", "enter"},
            {"Steve", "enter"},
            {"Martha", "exit"},
            {"Jennifer", "enter"},
            {"Joe", "enter"},
            {"Curtis", "exit"},
            {"Curtis", "enter"},
            {"Joe", "exit"},
```

```
                {"Martha", "enter"},
                {"Martha", "exit"},
                {"Jennifer", "exit"},
                {"Joe", "enter"},
                {"Joe", "enter"},
                {"Martha", "exit"},
                {"Joe", "exit"},
                {"Joe", "exit"}

        };
String[][] records1={
        {"Paul", "enter"},
        {"Paul", "exit"}
};

String[][] records2={
        {"Paul", "enter"},
        {"Paul", "enter"},
        {"Paul", "exit"},
        {"Paul", "exit"}
};

        String[][] records3={
                {"Raj", "enter"},
                {"Paul", "enter"},
                {"Paul", "exit"},
                {"Paul", "exit"},
                {"Paul", "enter"},
                {"Raj","enter"}
        };



        System.out.println(listOfEmployees(records));
         System.out.println(listOfEmployees(records1));
      System.out.println(listOfEmployees(records2));
     System.out.println(listOfEmployees(records3));
    }
}
```

Question 4:

You are working on a logic game made up of a series of puzzles. The first type of puzzle you settle on is "sub-Sudoku", a game where the player has to position the numbers 1..N on an NN matrix.

Your job is to write a function that, given an NxN matrix, returns true if every row and column contains the numbers 1..N

The UI for the game does not do any validation on the numbers the player enters, so the matrix can contain any signed integer.

grid1 = [[2, 3, 1],
[1, 2, 3],
[3, 1, 2]]
numbers 1,2,3)
-> True (A grid of size 3: every row and column contains the

grid2 = [[1, 2, 3],
[3, 2, 1],
[3, 1, 2]]
contain the numbers 1,2 and 3.
-> False (The first column is missing the value 2. It should
Similarly, the second column is missing the value

Question 5:

```
/*
You are with your friends in a castle, where there are multiple rooms named after flowers. Some of the rooms contain treasures - we call them the treasure rooms.

Each room contains a single instruction that tells you which room to go to next.

 *** instructions_1 and treasure_rooms_1 ***

 lily* ---------      daisy  sunflower
                |        |     |
                v        v     v
 jasmin --> tulip*      violet* ----> rose* -->
            ^    |      ^             ^       |
            |    |      |             |       |
            ------    iris            ---------

* denotes a treasure room, e.g., rose is a treasure room, but jasmin isn't.

This is given as a list of pairs of (source_room, destination_room)

instructions_1 = [
    ["jasmin", "tulip"],
    ["lily", "tulip"],
    ["tulip", "tulip"],
    ["rose", "rose"],
    ["violet", "rose"],
    ["sunflower", "violet"],
    ["daisy", "violet"],
    ["iris", "violet"]
```

```
]

treasure_rooms_1 = ["lily", "tulip", "violet", "rose"]

Write a function that takes two parameters as input:
* a list containing the treasure rooms, and
* a list of instructions represented as pairs of (source_room,
destination_room)

and returns a collection of all the rooms that satisfy the following two
conditions:

* at least two *other* rooms have instructions pointing to this room
* this room's instruction immediately points to a treasure room

filter_rooms(instructions_1, treasure_rooms_1) => ["tulip", "violet"]
* tulip can be accessed from rooms lily and jasmin. Tulip's instruction
points to a treasure room (tulip itself)
* violet can be accessed from daisy, sunflower and iris. Violet's instruction
points to a treasure room (rose)

Additional inputs

treasure_rooms_2 = ["lily", "jasmin", "violet"]

filter_rooms(instructions_1, treasure_rooms_2) => []
* none of the rooms reachable from tulip or violet are treasure rooms

 *** instructions_2 and treasure_rooms_3 ***

```

```
*** instructions_2 and treasure_rooms_3 ***

lily ---------          --------
             |          |      |
             v          v      |
jasmin --> tulip ---> violet*--^

instructions_2 = [
    ["jasmin", "tulip"],
    ["lily", "tulip"],
    ["tulip", "violet"],
    ["violet", "violet"]
]

treasure_rooms_3 = ["violet"]

filter_rooms(instructions_2, treasure_rooms_3) => [tulip]
* tulip can be accessed from rooms lily and jasmin. Tulip's instruction
points to a treasure room (violet)

All the test cases:
filter_rooms(instructions_1, treasure_rooms_1)    => ["tulip", "violet"]
filter_rooms(instructions_1, treasure_rooms_2)    => []
filter_rooms(instructions_2, treasure_rooms_3)    => [tulip]

Complexity Analysis variables:
T: number of treasure rooms
I: number of instructions given
*/
```

```java
import java.util.*;

public class CastleTreasureFinder {

   public static List<String> filterRooms(String[][] instructions, String[]
treasureRoom) {
       // Map to hold the count of incoming instructions for each room

       List<String> treasureRooms= Arrays.asList(treasureRoom);
       Map<String, Integer> incomingCount = new HashMap<>();
       Map<String, String> immediateDestination = new HashMap<>();

       for (String[] instruction : instructions) {
           String source = instruction[0];
           String destination = instruction[1];
           immediateDestination.put(source, destination);
           if(!source.equals(destination))
               incomingCount.put(destination,
incomingCount.getOrDefault(destination, 0) + 1);
```

```
        }



        List<String> validRooms = new ArrayList<>();


        for (String room : incomingCount.keySet()) {

            if(incomingCount.containsKey(room)){
            if (incomingCount.get(room) >= 2) {
                if (treasureRooms.contains(immediateDestination.get(room))) {
                    validRooms.add(room);
                }
            }
            }
        }

        return validRooms;
    }

    public static void main(String[] args) {
       String[][] instructions1= {{"jasmin", "tulip"},
    {"lily","tulip"},
        {"tulip","tulip"},
        {"rose", "rose"},
     {"violet", "rose"},
     {"daisy", "violet"},
{"iris", "violet"}};

       String [][]  instruction2={
                {"jasmin","tulip"},
                {"lily","tulip"},
                {"tulip", "violet"},
                {"violet", "violet"}
       };

        String [] treasureRooms={"lily", "tulip", "violet", "rose"};
        String[] tressureRoom2={"lily","jasmine","violet"};
        String[] treasureRoom3= {"violet"};


        System.out.println(filterRooms(instructions1,treasureRooms));

        System.out.println(filterRooms(instructions1,tressureRoom2));

        System.out.println(filterRooms(instruction2,treasureRoom3));
    }
}
```

Question 6:

```
/*
We are developing a new board game, Thrilling Teleporters.

The board consists of consecutive squares from 0 to last_square, some of the spaces also contain Teleporters, which are given as comma delimited strings "from,to".

The game is played as follows:
  1. Each turn, the player rolls a die numbered from 1 to die_sides.
  2. The player moves forward the rolled number of squares.
  3. The player stops at last_square if they reach it.
  4. If the player finishes on a square with a teleporter, they are moved to where the teleporter points.

Note: Only one teleporter is followed per turn.

A sample board with last_square 20 the following teleporters might look like this conceptually.
teleporters1 = ["3,1", "4,2", "5,10"]

   +-----+
   v     |
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
      ^     |  |               ^
      +-----+  +---------------+

Write a function that returns a collection of the possible squares a player can move to in a single turn, given the following inputs:
- A collection of teleporter strings
- The number of sides on the die
- The square the player starts on
- The last square on the board

Example:
With a 6-sided die, starting at square 0 with a board ending at square 20 (as pictured above)

Rolling a 1, 2 or 6 have no teleporters so they end at that square.
Rolling a 3 goes to 1, rolling a 4 goes to 2, rolling a 5 goes to 10.
The possible outcomes are, in order of die roll, are [1, 2, 1, 2, 10, 6].
If we remove the duplicates, the answer is [1, 2, 10, 6]
destinations(teleporters1, 6, 0, 20) => [1, 2, 10, 6]

Additional Inputs:
teleporters2 = ["5,10", "6,22", "39,40", "40,49", "47,29"]
teleporters3 = ["6,18", "36,26", "41,21", "49,55", "54,52",
                "71,58", "74,77", "78,76", "80,73", "92,85"]
teleporters4 = ["97,93", "99,81", "36,33", "92,59", "17,3",
                "82,75", "4,1", "84,79", "54,4", "88,53",
                "91,37", "60,57", "61,7", "62,51", "31,19"]
teleporters5 = ["3,8", "8,9", "9,3"]

Complexity Variable:
B = size of the board
Note: The number of teleporters, T, and the size of the die, D, are bounded by B.

All Test Cases:
(output can be in any order)
                                "91,37", "60,57", "61,7", "62,51", "31,19"};
    String [] teleporters5 = {"3,8", "8,9", "9,3"};
  }
}
```