Second Challenges

1. Election Race
2. Drop Lowest
3. Bus Stop Locator
4. Groceries
5. Find a Path

PROGRAMS DO NOT HAVE TO DONE IN ORDER

1 correct = 70%

2 correct = 80%

3 correct = 90%

4 correct = 100%

5 correct = 110%

1.Election Race

Program Name: race.py ***Input File: race.dat***

You are on the student committee and have been selected to count the ballots for the new student representative. Originally the committee wanted to count ballots by hand, but you decided to have students fill out a form with the names of the candidates. Now you have to write a program that takes in the file with the ballots and declare who won.

Input

- Each line will contain the name of the candidate.

Output

Print out who won the race and the number of ballots per representative. Output is in the order in which the name of the candidates first appear in the input file.

Example Input File

```
John
Sam
Carly
Alexa
John
Alexa
Carly
John
Sam
...
```

Example Output To Screen

```
John: 20 votes.
Sam: 51 votes.
Carly: 35 votes.
Alexa: 32 votes.
Sam won the race!
```

SUBMIT **race**

2. Drop Lowest

Program Name: drop.py ***Input File: drop.dat***

Write a program that takes in a list of students and their assignments. Calculate the average score of each student and then drop their lowest assignment and recalculate.

Input

- Each line in the file will contain the name of the student and the scores of their assignments, each student will have at least 3 grades.

Output

- Print out the student's name
- Print out the average score of each student (rounded to one decimal place)
- Print out their average score of each student after dropping their lowest score (rounded to one decimal place)
- The order of the output is in the order that the names first appear in the input file.

Example Input File

```
John 93 87 89 91 67 75
Alexa 87 92 85 88 84 89
Sam 84 85 93 92 79 77
Carly 71 73 77 81 75 76
...
```

Example Output To Screen

```
John: 83.7, 87.0
Alexa: 87.5, 88.2
Sam: 85.0, 86.6
Carly: 75.5, 76.4
...
```

SUBMIT **drop**

3. Bus Stop Locator

Program Name: bus.py ***Input File: bus.dat***

Most cities have a transportation system. Here in Austin, the metro system, Cap Metro, has routes and bus numbers. Depending on the route and bus number there is a set of bus stops. For each bus stop find the bus and route numbers associated with it.

Input

- Each line in the file will contain the coordinates the route number, bus number, and stop name

Output

Print out the bus stop with their associated route and bus number

Example Input File

```
7 102 Dean_Keeton
7 12 Downtown
7 12 Mueller
7 102 Guadalupe
7 102 Downtown
10 221 Dean_Keeton
10 221 Airport
20 305 Downtown
12 7 Mueller
...
```

Help

Remember that a single route can have multiple bus numbers, so you should not assume one bus per route. Store the data in a way that allows more than one bus to be connected to the same route.

Also consider that a route and bus pair can be reversed (ex: Route 7 Bus 12 vs. Route 12 Bus 7) and they may both stop at the same location. Your program should treat these as different buses, and the stop should list both pairs.

Just because it looks like a printed list in the output, does not mean you have to store it like that.

Example Output To Screen

```
Dean_Keeton: [Route: 7, Bus: 102] [Route: 10, Bus: 221]
Downtown: [Route: 7, Bus: 12] [Route: 7, Bus: 102] [Route: 20, Bus: 305]
Mueller: [Route: 7, Bus: 12] [Route: 12, Bus: 7]
Guadalupe: [Route: 7, Bus: 102]
Airport: [Route: 10, Bus: 221]

...
```

SUBMIT **bus**

4. Groceries

Program Name: groceries.py   Input File: groceries.dat

You own a local restaurant and regularly purchase supplies from several nearby stores. Instead of paying immediately, each store keeps a running tab for you. At the end of the month, you must calculate how much you owe each store in order to report your expenses for tax purposes.

Input

- The first line will contain a single integer N that indicates the number of lines that correspond with a supply and their respective price.
- After N lines, each line will contain the name of the store and the supplies bought from them.

Output

The total owed to each store in the order provided in the file. Round your numbers to two decimal places.

Help

Use the first line of the file to store the set of grocery items (you may want to use the **range()** function to help with storing). After storing this information, use a **for in loop** to read and process the remaining lines of the file.

Example Input File

```
5
milk 3.49
eggs 4.25
bread 2.99
cheese 5.50
apple 0.75
MarketFresh milk eggs bread
BakeryPlus bread milk
DairyDepot eggs eggs milk cheese apple
…
```

Example Output To Screen

```
MarketFresh: $10.73
BakeryPlus: $6.48
DairyDepot: $18.24
...
```

SUBMIT   **groceries**

5. Find a Path

Program Name: path.py ***Input File: path.dat***

GPS devices run an algorithm to figure out the shortest path from the source to the destination. GPS devices are more complex since they have to take into account the speed limit, traffic lights, traffic, etc. Your program will read in coordinate points and try to find a path from the source "S" to the destination "D". Other coordinates will have integers denoting the time it takes to travel. Your program will only be able check the surrounding roads (coordinates that are 1 away and only horizontally and vertically, but not diagonally). Choose the coordinate that adds the least amount of time.

Input

- Each line will contain the coordinates (x, y), "S" for source, "D" for destination or a positive integer, **T**, for time

Constraints:
1<=T<=100

Output

Print out all the coordinates (including the source and destination) of the path and total time. If there is no path print out "NO PATH".

Help

This program is "greedy" since it chooses the *optimal* choice at each step (the coordinate with the least time). While writing your program consider:

- Coordinates may not exist in the file, therefore can't be traveled to.
- Revisiting coordinates can cause your program to get stuck in an infinite loop.
- It is possible that no valid path exists from the source to the destination.

Example Input File

```
0 0 S
1 0 3
2 0 2
3 0 9
4 0 1
5 0 D
2 1 1
3 1 2
3 2 1
4 2 1
5 2 1
5 1 1
```

Example Output To Screen

```
Path: [(0, 0), (1, 0), (2, 0), (2, 1), (3, 1), (3, 2), (4, 2), (5, 2),
(5, 1), (5, 0)]
Total time: 12
```

SUBMIT **path**