Fall 2016 COP 3223H Program #2: Election Season
Due date: Please consult WebCourses for your section

Objectives

1. To learn how to apply a while loop to solve a problem.
2. To learn how to use a for loop in conjunction with other language elements to solve a problem.
3. Use lists to store parallel sets of data to help solve a problem.

Problem A: Where are my staffers? (staff.py)

Both candidate A and candidate B have strong ties to New York City. Because of their New York connection, often times, they are brusque with their staffers. As a consequence, their staffers have decided to troll them. In the past, the candidates could find their staffers via GPS of their cellphones. The staffers have determined how to block this information. To find their staffers, both candidates have to text and ask them where they are. The staffers have decided that it would be more fun to answer their question with either "hot", "cold", or "same", depending on whether the candidate has gotten closer to them, farther from them, or the same distance.

For the purposes of this problem, New York City's streets are arranged as a grid of equally spaced out N-S streets and E-W streets, called streets and avenues, numbered from 1 to 100, inclusive. We assume that the staffer is located at a street corner and that the candidate moves from street corner to street corner. We define the distance between the street corner at street *a*, avenue *b* and the street corner at street *c* and avenue *d* to be $|a - c| + |b - d|$.

The beginning of your program should randomly generate a corner where the staffer is located with a street and avenue integer in between 1 and 100, inclusive. Then, your program should ask the user where the candidate is currently located (two integers, one for street, one for avenue). If this location is the same as where the staffer is, tell the user that they've already found the staffer. Otherwise, ask the candidate where they have moved to. After this, your program should print out "hot" if the candidate is closer to the staffer than he/she was at his/her prior location, "cold" if the candidate is farther from the staffer than he/she was at his/her prior location, and "same" if the candidate is the same distance from the staffer compare to his/her prior location.

When the candidate (user) finds his/her staffer, end the program stating both the location of the staffer and the number of turns/moves it took for the candidate to find the staffer.

Input Specification

All input values from the candidate will be integers in between 1 and 100, inclusive.

Output Specification

After each movement of the candidate that moves to a position where the staffer is not, output one of the following three lines:

```
hot
cold
same
```

depending on whether or not the candidate is closer, farther or the same distance away from his/her staffer as before.

When the candidate finds the staffer, output a single line of the following form:

```
The candidate found the staffer at street X, avenue Y in N
moves.
```

where X is the street number where the staffer is, Y is the column number where the staffer is and N is the number of moves (0 or greater, 0 for if the candidate is originally where the staffer is).

Sample Program Run #1

```
What street corner are you starting at?
12
What avenue corner are you starting at?
13
The candidate found the staffer at street 12, avenue 13 in 0
moves.
```

Sample Program Run #2

```
What street corner are you starting at?
50
What avenue corner are you starting at?
50
What street corner did you just move to?
25
What avenue corner did you just move to?
25
hot
What street corner did you just move to?
20
What avenue corner did you just move to?
31
cold
What street corner did you just move to?
1
What avenue corner did you just move to?
50
same
What street corner did you just move to?
1
What avenue corner did you just move to?
10
The candidate found the staffer at street 1, avenue 10 in 4
moves.
```

Problem B: What is my chance of winning? (win.py)

The US decides its presidential elections with a rather peculiar system called the electoral college. In practice, this means that each state has a number of electoral votes assigned to it and that the winner of the state takes ALL of those votes. For example, Florida is assigned 29 electoral votes. Thus, if a candidate wins by a margin of 51% to 49% of Florida voters, that candidate gets all 29 of Florida's electoral votes.

Polls can give some accurate probabilities of candidates winning a particular state. Putting these together with the number of electoral votes in a state, we can calculate the probability of a candidate winning an election.

The technique we will use to solve this problem is brute force: for *n* states, we will try all possible 2^n outcomes, calculating the probability of each. For a regular election in the United States, this algorithm isn't feasible since 2^{51} (Washington DC is gets 3 electoral votes) is quite a large number, but for this problem, I will only test your program on a hypothetical country with at most 10 states that uses the same system the US uses. In a future assignment we'll learn how to efficiently make this calculation for 50 states and DC, so long as the total number of electoral votes isn't astronomical. (Our efficient algorithm will take roughly 51 x 538 steps, since there are 538 electoral votes in the US.)

One Method to Solve the Problem

Since the method to solve this problem is typically outside the scope of an introductory programming course, I'll simply outline the method for you here and the challenge for you will be implementing it from my description.

The user will be entering state by state information - both the number of electoral votes for a state as well as the probability of the candidate in question winning the state. (Read in the former as an integer and the latter as a floating point number in between 0 and 1, inclusive.)

As the user enters the information, your program will do two things: (1) update a variable storing the total number of electoral votes, (2) update two lists of information storing parallel information - number of electoral votes and the probability of that situation.

In the beginning of the algorithm your two parallel lists, call them votes and probs will look like this:

```
votes--->[0]
probs--->[1]
```

since if we are considering no states, then the only possible outcome is 0 votes with a probability of 1.

Let's say that the first state has 12 electoral votes with the candidate having a 45% chance of winning the state. Our algorithm will take each outcome stored in the votes and probs lists, and create a new lists tempvotes and tempprobs from it. The way this will occur is that for each entry in the votes and probs list we will apply two possibilities: the chance the candidate loses that state and the chance the candidate wins that state. In each case, we figure out the probability of

the outcome by multiplying the old probability by the new probability. To figure out the electoral votes, what we do is add the new state to the old total in the case that the candidate wins the state, and we keep it the same if the candidate doesn't win the state. Thus, after processing the first state given in the example, our lists look as follows:

```
tempvotes--->[0,12]
tempprobs--->[.55,.45]
total 12
```

This indicates that our candidate has a 55% chance of having 0 electoral votes and a 45% chance of having 12 electoral votes. Also, the total variable indicates that there are a total of 12 electoral votes. Before we move onto reading in the information about the next state, we must reassign both of our lists with the following assignment statements:

```
votes = tempvotes
probs = tempprobs
```

These two statements DON'T copy all the values into the original list, rather, they are efficient and simply make the pointers for both votes and probs point to where tempvotes and tempprobs point. Now, let's say that the second state has 19 electoral votes and our candidate has a 60% chance of winning that state. At the very end of our second loop, after reassigning votes and probs, our picture should look like this:

```
votes--->[0,19,12,31]
probs--->[.22,.33,.18,.27]
total 31
```

To obtain the first set of entries, we multiplyed .55 by .4, the probability of losing the second state to yield .22. If we lose the second state and previously had 0 electoral votes, we'd still have 0 electoral votes. To get the second set of entries, we multiplied .55 by .6, the probability of winning the second state to get .33 and added 19, the number of electoral votes in the second state to the old number of electoral votes (0) we had. To get the third set of entries, we multiplied .45 by .4, the probability of losing the second state to yield .18. The number of electoral votes in this scenario is simply 12, the old total. Finally, to get the last set of entries, we multiplied .45 by .6, the probability of winning the second state to yield .27. The number of electoral votes in this scenario is simply the old number(12) plus the 19 for winning this state.

Let's say our country only had two states. To determine the winner, we can do a for loop through our lists. For each number of votes that is greater than 1/2 of the total, we will add up the corresponding probabilities. In this case, the candidate wins the election with either 19 or 31 electoral votes, so we can obtain the total probability by adding .33 to .27 to get .63. Keep in mind that as you test this program with more states, the chance of getting the same number of electoral votes listed multiple times increases. This simply indicates that there may be more than one combination of states that yields the same exact number of electoral votes. For the purposes of this algorithm, this doesn't really matter. As we loop through the lists, regardless of whether a value in the first list is a repeated value or not, all we care is if it's more than 1/2 the total. If it is, we add the corresponding probability. If it isn't, we do nothing.

Input Specification
The number of states will be in between 1 and 10, inclusive. The number of electoral votes per state will be in between 3 and 100, inclusive. The probabilities for the candidate winning each state will be in between 0 and 1 inclusive, to at most 3 decimal places.

Output Specification
Output a single line of the format:

```
The candidate has a X chance of winning the election.
```

where X is the appropriate probability in between 0 and 1, inclusive.

Sample Program Run #1

```
How states does your country have?
2
How many votes is state 1 worth?
12
What is the probability you will win state 1?
.45
How many votes is state 2 worth?
19
What is the probability you will win state 2?
.6

The candidate has a .63 chance of winning the election.
```

Sample Program Run #2

```
How states does your country have?
3
How many votes is state 1 worth?
15
What is the probability you will win state 1?
.7
How many votes is state 2 worth?
7
What is the probability you will win state 2?
.2
How many votes is state 2 worth?
8
What is the probability you will win state 2?
.8

The candidate has a .588 chance of winning the election.
```

Note: Due to rounding issues, the actual print out may be to more decimals, don't worry about that for the purposes of this homework assignment, as long as the value of what's printed is pretty close (within 10^{-9}) to the correct answer.

Deliverables
Four source files:

1) *staff.py*, for your solution to problem A
2) *win.py* for your solution to problem B

All files are to be submitted over WebCourses.

Restrictions
Although you may use other compilers and coding environments, your program must run in IDLE for Python 3.2.5.

Grading Details
Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

Note: As mentioned in class, the input specifications are there to help you. Those are the requirements I guarantee I'll stick to when testing your program. You don't need to check if the user enters values within those parameters.