Function Name: `beautyAndBrains`

Inputs:

1. *(double)* A vector of beauty ratings
2. *(double)* A vector of brains ratings
3. *(double)* A vector of candidate numbers

Outputs:

1. *(char)* The perfect date
2. *(double)* Their ratings

Function Description:

Playing the dating game is always a struggle, and Tech students, being both intelligent and attractive, always have the hardest time finding the perfect date because we want people just like us: beautiful and intelligent.

Given a vector of a several potential dates' beauty ratings (on a scale from 1-10) and several potential dates' intelligence ratings (on a scale from 1-10), determine who the ideal date would be. If the sum of a potential date's beauty and brains is greater than or equal to 15 out of 20, they are the perfect date and the following is saved to the first output:

`'The perfect date is candidate #<num>.'` where `<num>` is the candidate's number from the 3rd input.

If none of the candidates have a score of at least 15, then the first output should be the following:

`'No perfect date was found, however candidate #<num> has the highest score.'`

Notes:

- There will never be a tie for the perfect date
- All of the input vectors will be of the same length

Function Name: `loveMeTinder`

Inputs:

1. *(char)* A string of the potential date's name
2. *(double)* 1x3 vector of your preferred age range, inclusive, and maximum distance (miles) from your location
3. *(double)* 1x2 vector of the potential date's actual age and actual distance
4. *(char)* The name of the potential date's college or university
5. *(logical)* A true or false value indicating if the first photo is a selfie
6. *(char)* The first word of the potential date's biography

Outputs:

1. *(char)* A string indicating a swipe to the left, a swipe to the right, or a Super Like

Function Description:

It's finally here (or come and gone if you didn't start last weekend): Singles' Awareness Day (S.A.D.). You've celebrated over the past few year with multiple pints of ice cream and reruns of *Grey's Anatomy*, but this year you've decided to try something different. Tossing your slippers aside, you've downloaded the popular "social discovery" app, Tinder, and decided that this is the year you spend S.A.D. with someone else. But you won't settle for just anyone; you have standards!

Starting with the basics, you set your age preferences (a low and high age, inclusive) and distance range. These values are the first, second and third elements of the second input, respectively. You make some more qualitative observations next and decide that you aren't a big fan of selfies or quotes in biographies, but will accept up to one of those traits (but not both). Whether or not their profile picture is a selfie will be represented by a logical in the 5th input. Input 6 is the first word of their biography and will start with a double quotation mark (") if it is a quote. The person's actual age and distance from you will be represented by the first and second element of the third input.

If a potential date's school is 'Georgia Tech', then they are allowed to have both a selfie and a quote, but must still abide by the age and distance settings. You will "Super Like" their photo if they meet these conditions.

Finally, you're calling it all off if a potential date's school is u(sic)ga. No matter how well the other criteria matched, you'll go back to Netflix before dating a "dawg".

After analyzing all this criteria, you will output a simple string:

`'Swipe <left|right> on <name>''s picture'`

OR

`'Super Like <name>''s picture'`

Notes:

- A quote will start with a double quotation mark (ASCII code 34); i. e. `'"Once...'`
- u(sic)ga will only ever be written as `'u(sic)ga'`.This is the grammatically correct spelling.
- Georgia Tech may be referred to as `'GT'`, `'Georgia Tech'`, or `'Georgia Institute of Technology'`
- The test cases given **are not** exhaustive. Be sure to test all possible conditions!
- A swipe to the right indicates the acceptance of the potential date's profile, whereas a swipe to the left indicates rejection.

Function Name: `lotteryTickets`

Inputs:

1. *(double)* An array of the purchased lottery ticket numbers
2. *(double)* A vector of the winning numbers

Outputs:

1. *(char)* A string representing the winnings earned from playing the lottery

Function Description:

Now that Valentine's Day has passed, you might be feeling a little low on cash. To fix this problem, you decide to play PowerBall!

Given an array of doubles representing purchased lottery tickets and a vector representing the winning numbers, write a function to determine the prize money won from the purchased lottery tickets. **Each row of the array represents a separate ticket.** If you've never played PowerBall before, here are the basics of how the game works.

Each ticket has 6 numbers. The first 5 are drawn from a pool of white balls, and the 6th number, known as the "PowerBall," is drawn from a pool of red balls. There are several ways to win money by playing Powerball, as shown by the following table:

# of white balls matched	**Powerball matched?**	**Winnings**
5	Yes	Grand Prize
5	No	$1,000,000
4	Yes	$50,000
4	No	$100
3	Yes	$100
3	No	$7
2	Yes	$7
1	Yes	$4
0	Yes	$4

Click here if you are interested in more information about PowerBall gameplay.

The output string will be one of three options, as represented by the following table:

Winnings	**Output**
Grand Prize	`'Congratulations! You've won the Grand Prize!'`
Any non-zero amount (excluding the Grand Prize)	`'Congratulations! You've won <num> dollars!'`
Zero	`'Too bad. You did not win any money.'`

Except for the Grand Prize, the total winnings should be additive. For example, if the array contains one ticket worth $100 and another worth $7, the total winnings should be $107. However, if the Grand Prize is won on any ticket, all other winnings are disregarded, and the Grand Prize is the only prize earned.

Notes:

- The white balls can match those on the winning ticket **in any order**.
- **None** of the tickets will have a repeated number.
- The winning numbers will be formatted in the same style as one purchased ticket, with the first 5 numbers representing the white balls and the 6th representing the Powerball.
- Every purchased ticket possesses a potential value.

Hints:

- The `find()` and `any()` functions are your friends.
- Iteration is not at all necessary to complete this problem. Remember what you have learned about logical masking and properties of logical vectors/arrays.

Function Name: `lostThatLovinFeelin`

Inputs:

1. *(logical)* An MxN array representing where love can be found
2. *(double)* A 1x2 vector representing your location

Outputs:

1. *(char)* A string describing which direction you should go to find love

Function Description:

Valentine's Day has come and gone (unless you are super assiduous and are reading this before Sunday) and instead of watching Netflix and eating ice cream, you have decided to go out and find love.

In true Tech fashion, you are given an array of logicals which represents where you can find that lovin' feelin', and your location in the array. The array of logicals is of size MxN, with both M and N guaranteed to be greater than or equal to 2. True values in the array represent where you can find some lovin' feelin', false values represent desolate, love-lacking areas.

The location vector is of the form `[row, col]`, where row and col are guaranteed to be valid row and column indices of the first input. This is where you are 'located' in the array. Your current location obviously does not contain love, otherwise this problem would be silly!

You can move up, down, left, right, or in any diagonal direction to find love. You will output a string determining the cardinal direction in which you should move to find love. You should check for love in a clockwise direction starting with the North direction (directly above your current location), then checking Northeast, then East, and so on. Whichever is the first direction you find love in should be the one that you output. Your output should be of the form:

`'Head <direction> to find love.'`

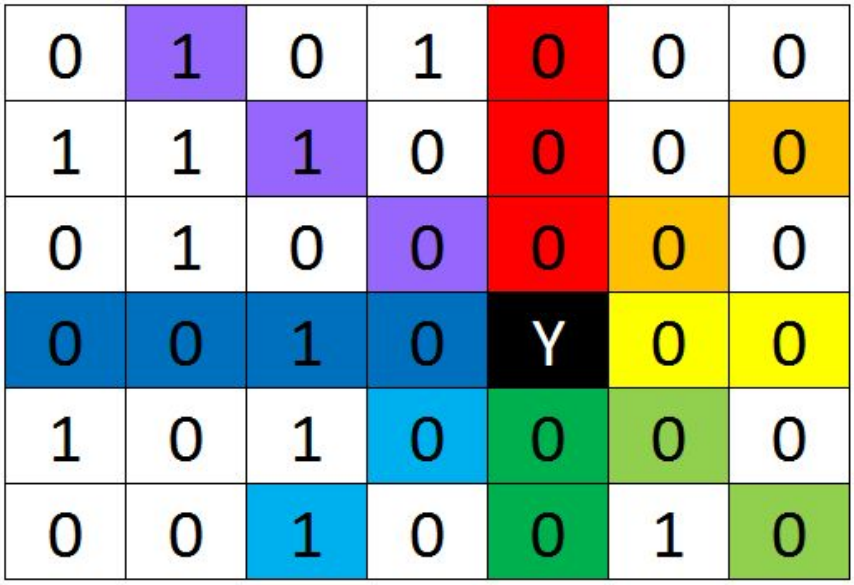
OR

`'There is no love to be found :('`

The second output option should only occur if no love is reachable from your current location. The possible directions are: `'North'`, `'Northeast'`, `'East'`, `'Southeast'`, `'South'`, `'Southwest'`, `'West'`, and `'Northwest'`.

An example is given on the next page.

Given a 6x7 love location array and a current location of `[4, 5]`, the locations you could reach are highlighted in the following visualization:



Each color corresponds to the cardinal direction of those indices. Red corresponds to North, orange to Northeast, Yellow to East, and so on. You would start looking for love in the North direction (the red indices) and discover that there is no love there. So you check to the Northeast and again find no love (**sigh**). Proceeding in this way, the first direction that contains love is the Southwest, so the output string would be:

```
'Head Southwest to find love.'
```

Notes:

- Be sure to use `isequal()` to check the formatting of your output string!
- Although the above example uses 1's and 0's for visual clarity, the actual input will be of class logical, not double.

Hints:

- The second input to the `diag()` function will be useful.
- If you are having trouble getting started, try just checking the vertical and horizontal directions, then add in the diagonals and lastly add in checking each cardinal direction separately.
- You will probably need to combine masking and indexing to solve this problem.