Function Name: `finalPredict`

Inputs:

1. (*double*) 5xN table of each team's points
2. (*char*) 5x5 array of team captains' names
3. (*char*) 1x5 string with the captain of the team you support

Outputs:

1. (*char*) A formatted string stating whether or not your team will make it to the finals

Topics: (*conditionals*), (*masking*), (*arrays*)

Background:

It is a matter of pride. The friend you once watched fool around with a kickball back in grade school recess has now become a professional kickball player. All those years of hard work, training, intense practice, and emotional distress during recess have led to this moment. Whether it was stopping the bullies from stealing your lunch or from messing with you on the playground, your friend has always had your back. Since this is such a big moment, it's understandable that you're anxious about how your friend's team is doing. You turn to your other friend who has always had your back... MATLAB!

Function Description:

Write a function that determines whether or not your favorite kickball team has the chance to make it to the finals or not. Each **row** in the first input represents a team's scores. The sum of these scores is the total score of that team this season. Based on the third input, you must check a specific row to see if the corresponding team's total points are greater than 100. To determine the correct row, compare the 3rd input to the **rows** of the 2nd input. Whichever row in the 2nd input that contains the name of the captain you support is the row of interest. The 2nd input will contain a randomized list of the following team captains' names: `'Rohan'`, `'Shayz'`, `'PaulJ'`, `'Ollie'`, and `'Divya'`. The specific team captain you support (3rd input) will always be one of these 5 captains' names. The points table (1st input) has the team's scores along the corresponding rows.

Output a different string depending on whether or not the team will make it to the finals.

- If the team has a total score less than 100, output:

```
'Better luck next year, <captain's name>'s team has only <total
                          score> points :('
```

- If the team has a total score of exactly 100, output:

```
'<captain's name>'s gotta work for their spot!'
```

- If the team has a total score greater than 100, output:

```
'Congratulations!! You're definitely going to see <captain's
                    name> and team in the finals!'
```

Example:

```
points = [ 20   15    3    4
           13   45    6    8
           90    4    7   36
           80   75   62    7
           77    5   13   45 ]
teams = 'Rohan'
        'Shayz'
        'PaulJ'
        'Ollie'
        'Divya'
capt = 'Ollie'

str = finalPredict(points, teams, capt)

str →
     'Congratulations!!  You're  definitely  going  to  see  Ollie
     and team in the finals!'
```

Notes:

- All numerical inputs are guaranteed to be positive integers.
- To add an apostrophe (`'`) in a string, place two single quotes consecutively.
    - For example: str = `'MATLAB''s the best!'`

Hints:

- `strcmp()` might be useful for finding the correct row to sum across.

Function Name: `majorLeagueStats`

Inputs:

1. (*char*) Player's name
2. (*double*) Value of player's strength
3. (*double*) Value of player's agility
4. (*double*) Value of player's speed
5. (char) Player's handedness

Outputs:

1. (*char*) A string determining whether or not a player is qualified to play in Nationals

Topics: (*if conditionals*), (*switch conditionals*), (*nested conditionals*)

Background:

You are the new head coach for the Atlanta Braves' renowned baseball team. Unfortunately, you're only allowed to take a certain amount to the Nationals this year. On top of that, there were a record number of players on the team this year, and you have no idea how to select your top players. However, luckily for you, you went to Georgia Tech and took 1371 back in the day. Time to put your MATLAB skills into use!

Function Description:

You're given information on a player's name (input 1), strength (input 2), agility (input 3), speed (input 4) and handedness (input 5). First calculate the players total speed stat by multiplying agility and speed. Then, determine the player's position. If the player's handedness is `'right'` and their total speed stat is greater than or equal to their total strength stat, their position is `'outfielder'`, otherwise it is `'batter'`. If a player's handedness is `'left'` and their total speed is less than their strength, their position is `'pitcher'`, otherwise, their position is `'batter'`. Finally, if the handedness is 'both', the player's position is `'pitcher'`.

Next, you will determine if the player will make it to Nationals. If the player is a batter they can make it to nationals if either their strength stat or total speed stat is at least 25000. If the player is an outfielder, they will make it to Nationals if their total speed stat is at least 50000. If the player is a pitcher, they will automatically make it to Nationals if their handedness is 'both', but they can also make it if their strength score is at least 100000. Finally, output a string detailing what happened to the player.

Output a different string depending on whether or not the player qualifies for Nationals.

- If the player qualifies, output:

```
'CONGRATULATIONS! <player's name> has been selected to play in
Nationals as a(n) <position>.'
```

- If the player does not qualify, output:

```
'<player's name> was not qualified enough to play in Nationals as
a(n) <position>. Try again next year!'
```

Example:

```
name = 'Lauren'
strength = 16360
agility = 200000
speed = 76
handedness = 'right'

str = majorLeagueStats(same, strength, agility, speed, handedness)

str →
     'CONGRATULATIONS! Lauren has been selected to play in Nationals
     as a(n) outfielder.'
```

Notes:

- Pay close attention to your conditional operators.
- In the output string, print the player's name exactly as it was given to you.

Function Name: `awardSeason`

Inputs:

1. (*char*) a string containing the name of Player A
2. (*char*) a string containing the name of Player B

Outputs:

1. (*char*) a sentence stating who won the Ballon d'Or award

Topics: (*switch conditionals*), (*nested conditionals*)

Background:

You are an esteemed writer for ESPN and have been matched to cover the odds of who is winning the Ballon d'Or based on particular matchups for the finalists. This year is a little strange. They are only doing two finalists, it isn't separated by gender, and there is someone that doesn't even play soccer in the running. You are a little too lazy to write up the headlines, so you decide to write a MATLAB function that will write up what's occurring for you.

Function Description:

Write a function that will take in two player name inputs (player A and player B) and output the desired string according to the matchup. The only possible inputs for this function are

```
'Ronaldo'
'Solo'
'Messi'
'Kane'
'Wambach'
'Paul Johnson'
```

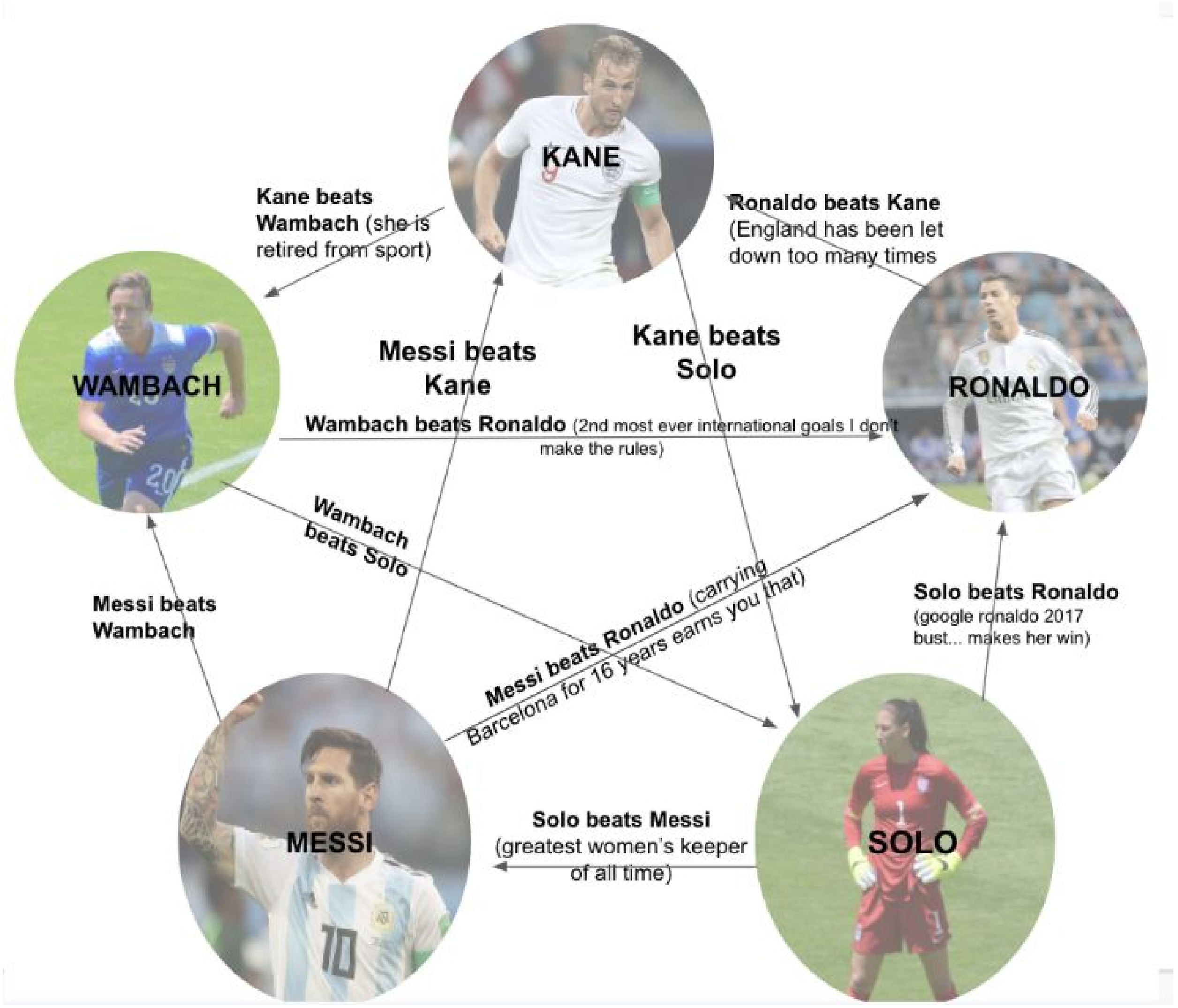
Format the output string depending on the outcome defined by the diagram below.

- If one of the players wins, output:

```
'<winner's name> wins the specialty Ballon d'Or over <loser's
name>! They destroyed the competition!'
```

- If player A and player B are the same person, output:

```
'This is just an ego boost for <player's name>. This isn't
really the Ballon d'Or.'
```


KANE
WAMBACH
RONALDO
MESSI
SOLO
Kane beats Wambach (she is retired from sport)
Ronaldo beats Kane (England has been let down too many times
Messi beats Kane
Kane beats Solo
Wambach beats Ronaldo (2nd most ever international goals I don't make the rules)
Wambach beats Solo
Messi beats Wambach
Messi beats Ronaldo (carrying Barcelona for 16 years earns you that)
Solo beats Ronaldo (google ronaldo 2017 bust... makes her win)
Solo beats Messi (greatest women's keeper of all time)

However, there is an exception! If at least one of the players is infamous Georgia Tech football coach Paul Johnson, he will win the award no matter what. If `'Paul Johnson'` is one or both of the players, output the string:

```
'Paul Johnson does it again! He has won against the odds!'
```

Example:

```
playerA = 'Messi'
playerB = 'Solo'

str = awardSeason(playerA, playerB)

str →
     'Solo wins the specialty Ballon d''Or over Messi! They
     destroyed the competition!'
```

Function Name: `passItOn`

Inputs:

1. (*double*) A MxN array representing a hockey rink

Outputs:

1. (*char*) A string describing your play and its chance of success

Topics: (*conditionals*), (*indexing*), (*masking*), (*arrays*)

Background:

The year is 2021 and the International Ice Hockey Federation has just released its new set of rules regarding the number of players each team is allowed to bring to the rink. To spice the game up a bit, the Federation has determined that one team will be allowed an unlimited number of players on the rink at the same time, while the other team is limited to just two. To ensure fairness, however, they have determined that the disadvantaged team will gain access to a wonderful tool for determining how likely the team is to succeed on any given play. That tool: MATLAB!

Function Description:

You are given an array of doubles representing the hockey rink, which is structured as follows:

- Zeros represent empty spots
- You are represented by the number 10
- Your teammate is represented by a **positive** integer
- Players on the opposite team are represented by **negative** integers

The only two players on your team are yourself and the teammate represented by the other positive integer, but there are an undetermined number of players on the rink from the opposite team. It is your job to build a string describing how you intend to pass the puck to your teammate, and the chance of the pass succeeding. To do this, follow these steps:

1. Using your and your teammate's positions, determine the direction you will be passing the puck. If your teammate is directly above, below, to the right, or to the left of you in the array, the direction will be `'north'`, `'south'`, `'east'`, or `'west'` respectively. If they are not located in the same row or column as you, the direction would be `'northeast'`, `'northwest'`, `'southeast'`, or `'southwest'`.

2. Index out the subarray whose corners are the positions of you and your teammate. For example:

```
[ 0   0   0   0   0   0
  0   0   0   0   8   0                          [ 0   0   0   8
  0   0  -5   0   0   0          →                 0  -5   0   0
  0   0   0  -6   0   0                            0   0  -6   0
  0  10   0   0   0   0                           10   0   0   0 ]
  0   0   0   0  -4   0 ]
```

The sum of all the positive integers in this subarray will represent your team's cumulative rating. The absolute value of the sum of all the negative integers in the subarray will represent the opposing team's cumulative rating. In this example, your team has a rating of 18 and the opposing team has a rating of 11.

3. Output the final string as follows:

    - If your team's cumulative rating is higher than the opposing team's cumulative rating by 10 or more, output:
      `'I shot the puck to the <direction> with guaranteed success.'`
    - If your team's cumulative rating is higher than the opposing team's cumulative rating by less than 10, multiply the difference in rating by 10 to determine your chance of success as a percentage. Output:
      `'I shot the puck to the <direction> with a <percent chance of success>% chance of success.'`
    - If your team's cumulative rating is equal to or lower than the opposing team's rating, output:
      `'I chose not to shoot the puck to the <direction>.'`

Example:

```
hockeyRink = [ 0   0   0   0   0   0
               0   0   0   0   8   0
               0   0  -5   0   0   0
               0   0   0  -6   0   0
               0  10   0   0   0   0
               0   0   0   0  -4   0 ]

play = passItOn(hockeyRink)

play →
     'I shot the puck to the northeast with a 70% chance of
     success.'
```

Notes:

- You are guaranteed to have at least one player of the opposing team in the subarray.
- It does not matter if there is an opposing player between you and your teammate, the probability of the shot is **only** determined by the difference in teams' cumulative ratings.
- Your teammate's number is guaranteed to not equal 10.
- Similar to apostrophes in strings, use a double percent sign (`'100%%' → 100%`) to get a percent sign in a string.

Function Name: `geoffCollins`

Inputs:

1. (*char*) A 5x10 character array representing a football field and associated players

Outputs:

1. (*char*) An updated 5x10 character array representing the field, players, and path of the football after the play is run
2. *(char)* A statement describing the outcome of the play

Topics: (*control flow*), (*nested conditionals*), (*masking*), (*RHS and LHS indexing*)

Background:

4th quarter. 12 seconds left on the clock. It's 4th and goal for the College Football Playoff National Championship against Alabama. Geoff scans his playbook one last time and then looks up, a mischievous twinkle in his eyes. He knows what to do.

JK. It's 4th and 26, halfway down the field, and Tech is facing the Citadel, hoping to pull out a single win for the year. Geoff pulls his 404 The Culture™ cap further over his eyes and crosses his arms over his playbook. No amount of preparation can save the Jackets now. He summons the spirit of Paul Johnson, smacks his quarterback's bum as the offense takes the field, and crosses his fingers.

Function Description:

You are given a football field represented by a 5x10 character array. The quarterback (the starting position of the football) is represented by the character `'Q'`. Any receivers that exist will be represented by the character `'R'`. Any defenders that exist will be represented by the character `'D'`. Any open spaces will be represented by the space character. The `'Q'` is guaranteed to be in the first row of the array. The objective is to move the ball to the bottom row of the array.

There are two potential plays you can make, **prioritized in this order**:

1. The quarterback can throw the ball to a receiver. The receiver must be located *EXACTLY* 4 indices away from the quarterback, exclusive, either straight down, towards the bottom-left diagonal, or towards the bottom-right diagonal. If a receiver is located in one of these positions, the play is **successful**. However, if one or more defenders are located directly adjacent to the **receiver**, not including diagonals (left one index, right one index, or "down" one index (up the array)), the play is **blocked**. If no receivers are located exactly 4 indices away, this play **cannot be used**.
2. The quarterback can run straight down the field. The quarterback moves down the column he was originally located in until reaching the bottom row or encountering a defender. If he reaches the bottom row, the play is **successful**. If a defender is located in the quarterback's path once he begins running, the play is **blocked**. If a defender is

located in the index position directly below the starting position of the quarterback, this play **cannot be used**.

If you are **able** to use the throwing play, you must use it, even if the play will be blocked later on. For example, if you are able to throw the ball, you must do so, even if a defender is located next to the receiver - you may not run the ball, even if this play would be successful.

If both of the plays **cannot be used** (no receivers are in the correct range and the quarterback cannot run straight down at all), the **play fails**.

As the play progresses, the path of the football must be marked. Including the starting index of the quarterback, change each index that the ball passes through to an `'X'`. If the ball is thrown, these indices may be open spaces, defenders, or other receivers. If the ball is thrown and blocked, still change the path the ball would take to `'X'`. If the ball is run and blocked, change all of the empty spaces between the quarterback (inclusive) and the defender (exclusive) to an `'X'`. The defender that blocks the play should not be changed to an `'X'`, but the quarterback should be. If the **play fails**, still change the quarterback starting position to an `'X'`. Output the updated character array. Additionally, output a statement describing the outcome of the play.

- If the play does not fail, output:

  `'The quarterback <ran/threw> the ball and the play was <successful/blocked>.'`

- If the play fails, output:

  `'Even Paul Johnson can't face impossible odds.'`

Examples:

```
field1 =   '    Q     '
           '    D     '
           '          '
           '    R     '
           '       DR '

[out1, sent1] = geoffCollins(field1)

out1 →     '    X     '
           '    DX    '
           '      X   '
           '    R  X  '
           '       DX '
```

```
sent1 →
     'The quarterback threw the ball and the play was blocked.'


field2 =   '    DQ    '
           '          '
           '          '
           '      R   '
           '    D D   '

[out2, sent2] = geoffCollins(field2)

out2 →     '    DX    '
           '     X    '
           '     X    '
           '     XR   '
           '    DXD   '

sent2 →
     'The quarterback ran the ball and the play was successful.'
```

Notes:

- If there are multiple receivers on the field, a **maximum** of one is guaranteed to be located at the correct throwing range.
- You do not have to worry about indexing out of bounds when checking for the locations of the potential receivers, or the defenders potentially blocking the receivers.
- You do not have to worry about changing a defender potentially blocking a receiver to an `'X'` before checking if the play was successful or blocked (this would only occur if the ball was thrown straight down and a defender was located exactly above the receiver). This situation will not be included.
- A receiver will never block the path of a running quarterback.

Hints:

- The `find()` function may be useful to find the starting location of the quarterback.
- Try to create your own scenarios and test them against the solution code to make sure your code passes more than just the three given test cases!

Extra Credit

Function Name: `romanHoops`

Inputs:
1. (*char*) The message you need to decrypt
2. (*char*) The keyword needed to decrypt the message
3. (*double*) The key shift used to encrypt the keyword

Outputs:
1. (*char*) The decrypted message
2. (*char*) The decrypted keyword

Banned Functions:

`upper, lower, unique, sort`

Topics: (*indexing*), (*conditionals*)

Background:

The time has come again. The Georgia Tech Yellow Jackets are preparing to face off against their most hated rival in a feud going back over a century. One that the Yellow Jackets are proud to continue. The call to the fans is "What's the good word?" and both players and the crowd roar back "To Hell with Georgia!" You know that u[sic]GA will try to win with whatever means necessary and as such you have made sure to encrypt all the GT gameplans ahead of time. You finish just in time and the Yellow Jackets swarm onto the basketball court!

Wait, what sport did you think I was talking about?

Function Description:

The eponymous caesar cipher is one of the most widely known encryption techniques. To use it you simply shift the alphabet left by a certain number of characters, wrapping back around to the end, and then replace all the letters with their new equivalents. A shift of 1 would see A become B, B become C, and so on until Z becomes A. A shift of 2 would have A become C, B become D, and so on until Z becomes B.

Unfortunately, this cipher is easily decrypted as there are only 25 possible ways to encrypt a message with it. (Replacing A with A does not actually encrypt a message.) As such there have been multiple variations of the caesar cipher developed, many of which use a keyword in addition to a shift number, including the one you will implement in MATLAB.

Write a function that decrypts an encrypted message (*input 1*). Given the encrypted keyword (*input 2*), perform the following steps to obtain the decrypted keyword.
1. Make all letters lowercase and remove all punctuation, numbers, and spaces.
2. Remove repeated letters.
3. Perform a simple caesar cipher with the given shift number (*input 3*).

Next, declare two new variables which we will call `shiftA` and `shiftB`. Assign the positive distance between the first and last characters to `shiftA`. Assign the positive distance between the second and second-to-last characters to `shiftB`. For example, if your decrypted keyword is:

```
decKeyword = 'hellothere'
```

then:

```
shiftA → 3
shiftB → 13
```

If `shiftA` ends up equaling zero, use the value 1 for `shiftA`. Likewise, if `shiftB` ends up equalling zero, use the value 2 for `shiftB`.

Depending on the most common letter in the decrypted keyword, each shift number will be applied to different parts of the original message in another simple caesar cipher. If the most common letter in the resulting decrypted keyword is...

- ... a consonant and an odd alphabetical character (c is #3 in the alphabet, so it is odd) use `shiftA` on the odd indices of the encrypted message and `shiftB` on the even indices.
- ... a consonant and an even alphabetical character (b is #2 in the alphabet, so it is even), apply `shiftA` to every odd character in the encrypted message and `shiftB` on the even ones. In this case, if one shift number is even and the other odd, double the odd one.
- ... a vowel (not including y), the first half of the encrypted message will be decrypted with `shiftA`, and the second half of the encrypted message will be decrypted with `shiftB`. Should the message be of odd length, the halfway point is determined by whichever character next to the middle one is larger, with the corresponding side containing the middle character. For example, if the message is:

  ```
  msg = 'gibberish'
  ```

  Compare the `b` to the `r`. Since `r` is "greater" than `b`, the `e` goes to the second half:

  ```
  firstHalf → 'gibb'
  secondHalf →  'erish'
  ```

  Do not forget to recombine the two halves of the message.

The resulting string, once these operations are performed, is your decrypted message. Output this along with the decrypted keyword.

(continued...)

Example:

```
encMsg = 'Tfobuvt Qpqvmaywak Xusgtay'
key = 'U!rr pp.352453xxox.xvv'
shift = 3

[decMsg, key] = romanHoops(encMsg, key, shift)

key →
     'romulus'

decMsg →
     'Senatus Populusque Romanus'
```

Notes:

- Do not change any of the capitalization of punctuation when performing a caesar cipher
- A and C are odd alphabetical characters, while B is even. B is larger than A while smaller than C. These pattern/trend hold throughout the alphabet
- The shift number will never be negative
- Every keyword will have at least 4 unique letters
- There will never be a tie between which letters in the keyword are the most common
- Letters in the keyword will never repeat more than once