Function Name: `ERS`

Inputs:

1. *(double)* A 1xN array representing a stack of cards

Outputs:

1. *(char)* A sentence indicating if the stack should be slapped

Function Description:

Egyptian Ratscrew (ERS) is a popular card game where players take turns laying down cards in a stack and must be the first to slap the stack when they see certain combinations in order to the collect the cards, with the goal of collecting the entire deck. It is not necessary to have played the game to understand the problem, but if you would like more information you can read about it here.

Given a stack of cards, represented as an array of doubles, determine whether the stack should be slapped. The input array will go from bottom-up, that is, the first element is the bottom card of the stack, and the last element is the top card visible to the players. Face cards (Ace, Jack, Queen, and King) will be represented by number values (`1`, `11`, `12`, and `13`, respectively). For example, the input `[8, 12, 1, 2, 2, 13]` represents Eight, Queen, Ace, Two, Two, and King, where the King is on the top.

There are many combinations people use when playing, but they must be agreed on before the game, so for our purposes we will focus on four: sandwiches, doubles, marriages, and sevens. The table below provides a description of each:

Sandwich	Two cards of the same value separated by only one card of a different value is a sandwich, for example `[5, 10, 5]`. If there is more than one card in between, such as `[5, 10, 8, 5]`, it is called a Big Mac and cannot be slapped.
Double	Two cards of the same value right after eachother is a double and should be slapped, such as `[4, 4]`.
Marriage	Many groups also slap on marriages, which is a King and a Queen right after eachother. `[12, 13]` and `[13, 12]` are both considered marriages.
Seven	Though not very common, some groups 'slap on seven', meaning you can slap the deck if there is a `[7]` on top regardless of the previous card!

The output will be a string stating whether the stack should be slapped, and if so, which combination is present. If the stack should not be slapped, your output should be:

```
Do not slap, it is a trap!
```

If there is a slappable combination on top, your output should be:

```
There is a <combination>, so quickly slap!
```

For example, if the input is [11, 3, 1, 3] your output should be

```
There is a sandwich, so quickly slap!
```

Notes:

- The input will have at least one element, but otherwise can be of any length.
- There will never be two slappable combinations.
- The combination should always be printed in all lowercase (`'marriage'`, not `'Marriage'`, etc.)
- The stack cannot be slapped for a combination found in the middle of the stack, only combinations at the very top. So if the input is `[8, 3, 12, 4, 12, 2]` , the stack should not be slapped even though there is a sandwich in the middle.
- The suit of the card is irrelevant.

Function Name: `checkCollision`

Inputs:

1. *(double)* An MxN array representing a snake game board
2. *(char)* A letter indicating the direction of the next move

Outputs:

1. *(char)* A string describing the outcome after the next move

Function Description:

You find yourself very bored in your room one day, having completed the week's CS1371 homework so early. What better way to entertain yourself than writing the classic game, snake, in MATLAB!

`checkCollision` is the first function you have to write in order to complete this game. Given an array representing the current game board, and a letter indicating the direction of the next move, you have to determine the outcome of the game after the move is made.

The board will be composed of three types of elements: empty space (`0`), the snake (`>0`), and a cookie byte (`-1`), the snake's food. Below is a visualization of a potential game board.

```
board = [0  0  0  0  0  0  0;...
         0  1  2  3  4  0  0;...
         0  0  0  0  5  0  0;...
         0  0  0  0  6  0  0;...
         0  0  0  0  7  8  9;...
         0  0  0  0  0  0 -1;...
         0  0  0  0  0  0  0]
```

The snake will always be represented as consecutive integers starting at 1. The head of the snake is always the largest value on the game board.

The direction of the snake's next move is determined by the second input, where the character `'N'`, `'W'`, `'E'` or `'S'` will indicate one step towards the North, West, East or South, respectively. Based on the next move, there could be four possible outcomes and four corresponding output phrases:

1) If the snake moves into a free space (`'N'` in the above example), the output should be
   `'continue'`
2) If the snake reaches a cookie byte (`'S'` in the above example), the output should be
   `'Cookie! Have a byte!'`
3) If the snake collides with a wall (`'E'` in the above example), the output should be
   `'Oh no! Firewall!'`
4) If the snake collides with itself (`'W'` in the above example), the output should be
   `'You hit yourself. Stack overflow.'`

Notes:

- The length of the snake is guaranteed to be at least 2.
- There is guaranteed to be one and only one cookie byte on the board.

Function Name: `rpsls`

Inputs:

1. *(char)* Player 1's move
2. *(char)* Player 2's move

Outputs:

1. *(char)* A string describing who won

Function Description:

Let's play a game of rock-paper-scissors-lizard-spock! Given the moves of two players, write a function called `rpsls` that determines who won the game. If you are unfamiliar with The Big Bang Theory, RPSLS is a variant of Rock-Paper-Scissors. Instead of just 3 options, each player can throw a `'rock'`, `'paper'`, `'scissors'`, `'lizard'`, or `'spock'`. The rules for what beats what are as follows:

1. Rock beats scissors and lizard
2. Paper beats rock and Spock
3. Scissors beats paper and lizard
4. Lizard beats paper and Spock
5. Spock beats rock and scissors

Based on these rules, the function should output one of the following strings:

`'Player 1 wins!'` if player 1 wins
`'Player 2 wins!'` if player 2 wins
`'It's a tie!'` if both players choose the same move

More information on RPSLS can be found here.

Notes:

- You are guaranteed that both of the inputs will be lowercase and one of the 5 possible moves.

Function Name: `yahtzee`

Inputs:

1. *(double)* A 1x5 vector of integers from 1 to 6, inclusive

Outputs:

1. *(double)* The maximum number of points possible
2. *(char)* A string recommending the corresponding dice combination

Function Description:

You and your roommates love spending your Friday nights playing Yahtzee, the addicting dice game in which players roll five dice and collect points based on different scoring combinations. Given a vector representing dice, write a function to determine which combination will score the most points.

The dice combinations are as follows:

Combination	Description	Points
`'3 of a Kind'`	One number appears 3 times.	Sum of all dice
`'4 of a Kind'`	One number appears 4 times.	Sum of all dice
`'Full House'`	3 of one number and 2 of another number	25
`'Small Straight'`	4 consecutive numbers	30
`'Large Straight'`	5 consecutive numbers	40
`'Yahtzee'`	All five dice are the same number	50
`'Chance'`	Any other sequence	Sum of all dice

Determine which combination(s) your dice fall under. If you dice meet multiple criteria, you need to determine which combination scores the most points. Your first output should be the maximum score for the given input. Your second output should be a string with the following format identifying which combination is being used to score the points:

`'I have a <combination>!'`

If your combination is a Chance, you must have had bad luck rolling your dice. Replace the `'!'` with a space and a frowning face (`' :('`).

However, if your combination is a Yahtzee, your entire second output should simply be:

`'YAHTZEE!'`

Notes:

- While `'Small Straight'` and `'Large Straight'` require sequential numbers to be present, these numbers may not be in order. For instance, the vector `[4 1 3 5 2]` is a `'Large Straight'`.

Hints:

- You may find the `sort()` and `diff()` functions useful.
- Be careful about the order of your conditionals.