Function name: <yourFirstName>_<yourLastName>_scrabbleShuffle

Example: kantwon_rogers_scrabbleShuffle

Inputs (3):

- (char) an array representing the 10x10 grid of scrabble tiles before shuffling
- (double) number of players (at least 2)
- (logical) true or false on if a 2-2-3-3 shuffle is also done.

Outputs (7):

- (char) an array representing the 10x10 grid of scrabble tiles after they have been shuffled **for your last simulated run**
- (double) Average number of scrabble tiles located in the same spot as if it weren't shuffled
- (double) Average number of scrabble tiles that are located next to their alphabetical neighbor
- (double) Average number of 2 of the same scrabble tiles next to each other
- (double) Average number of 3 of the same scrabble tiles next to each other
- (double) Average number of 4 of the same scrabble tiles next to each other
- (double) Average number of 5 of the same scrabble tiles next to each other

Disclaimer: THIS FUNCTION SEEMS DIFFICULT BECAUSE I HAVE A LOT OF WORDS DESCRIBING EVERYTHING, BUT IT ACTUALLY ISN'T TOO BAD. IT IS JUST ARRAY MANIPULATIONS!!

Clarifications to the problem given other students' questions/solutions are made in green

Function Description:

Matt Parker, a mathematician, posted a video that proposes a way that people can play scrabble together over video conferencing. Of course, the main challenge with making this work is that in real scrabble, everyone is choosing from the same 100 tiles that are in a bag and when someone chooses a tile only they know what they got. Moreover, if Player 1 chooses a tile, that tile should also not be able to be chosen by any other player; however, the other players can't know exactly what they cannot choose or they would know what Player 1 chose! So in this video HERE he describes an idea that he believes will allow people to randomly mix up scrabble tiles in the same way and have all parties involve not know how things are mixed. **He claims that the method he comes up with is just as random as choosing tiles from a bag. This is what you will be testing!**

In this function you will write code that mimics the scrabble shuffling described in the video. The goal is to then test if this shuffling matches the numbers he had computed in order to describe the "randomness" of the shuffle when averaged across 10,000,000 simulations. He starts with a 10x10 grid with all of the scrabble tiles arranged in alphabetical order across the rows like:

A	A	A	A	A	A	A	A	A	B
B	C	C	D	D	D	D	E	E	E
E	E	E	E	E	E	E	E	E	F
F	G	G	G	H	H	I	I	I	I
I	I	I	I	I	J	K	L	L	L
L	M	M	N	N	N	N	N	N	O
O	O	O	O	O	O	O	P	P	Q
R	R	R	R	R	R	S	S	S	S
T	T	T	T	T	T	U	U	U	U
V	V	W	W	X	Y	Y	Z	"	"

I will provide you a .mat file in this folder that has this starting grid

He said that given 10,000,000 simulations of randomly drawing all 100 tiles and placing them in a 10x10 grid you can expect on average:

- Fixed points: 5.9604827
  - This means that about 6 tiles will be placed in the same spots as the unshuffled grid
- Alphabetical neighbours: 6.6008077
  - This means that tiles are next to another tile that comes directly before or after it in the alphabet (this includes wrapping down a row back to the first column)
- Pairs: 4.3016657
  - This means that 2 of the same letters are next to each other (this includes wrapping down a row back to the first column!)
- Triples: 0.2712588
  - This means that 3 of the same letters are next to each other (this includes wrapping down a row back to the first column!)
- Fours: 0.0184608
  - This means that 4 of the same letters are next to each other (this includes wrapping down a row back to the first column!)
- Fives: 0.0012537
  - This means that 5 of the same letters are next to each other (this includes wrapping down a row back to the first column!)

This is our base assumption (however, you can also write some code if you want to actually test this).

Having looked at the numbers from people who have already completed it, it seems as if in Matt Parks solution he DOES NOT include a smaller category in a larger one. What I mean by that is if you have a set of four of the same letter 'AAAA' that also includes 3 pairs and 2 triples. I think his code counts this as a set of 4 and no pairs or triples. You can decide how you want to do it, but if you do include smaller categories in larger ones, then your average numbers will be larger than his!

Description of Shunting

In the video he then goes on the describe his method of shuffling the board. He describes what is called a "shunt" which means shifting a column/row of the grid down/right and then wrapping the letters that would have fallen off back up to the top/left. In order to randomly decide how all of the shunting happens, players randomly choose a 10 digit number and each digit says how much each column/row should be shifted. Each player involved should choose a 10 digit number. Given multiple players the first player's number represents the first round of shunting **columns**. Then the 2nd player's number represents the second round of shunting **rows**. Then the 3rd players number represents the third round of shunting **columns.** So essentially, the numbers from odd players represent shunting columns, and the numbers of even players representing shunting rows. However, the order of shunting should go from player 1 to player N. The second input to this function takes in the number of players in the game. You should then randomly generate (possibly in a vector?) that quantity of 10-digit numbers.

For example, if my 2nd input was 3 my function could randomly create a vector of three 10-digit numbers like [1234567890, 0987654321,1357924680]. First I should shunt the **columns** as described in the number 1234567890 (shift the first column down by 1, then the 2nd column down by 2..etc). Then I should shunt the **rows** as described by the 2nd number 0987654321 (shift the first row to the right by 0, shift the second row to the righty 9…etc). Finally, I should shunt the **columns** based on the third number 1357924680.

Description of 2-2-3-3

After shunting your function should then execute or not execute a 2-2-3-3 shuffle. The 3rd input of the function dictates if this should happen or not.

I really don't think trying to describe the 2-2-3-3 shuffle in printed words is the best way to go about this. Instead, refer to **minute 12** of the video and that is where the description of the process is explained.

Test Cases/HOW DO I WIN?!

I don't have a solution written, but I know how each step could be coded. That being said, your submission for this challenge should be code that you wrote that does this whole process and it should be **commented well enough so that I can easily follow your thought process**. (But you don't have to have an insane amount of comments). Ideally, if you go through the process in the same way described in the video, your numbers will be very similar to the numbers he reports in the description of the video.

I'm not providing any test cases. I can't. Your function will be generating random numbers. However, what I recommend doing is first trying to solve the problem as if the shunting numbers weren't randomly generated. For instance, I created those 3 shunting values above and given those exact 3 numbers, I should know exactly how the board will be arranged. I would suggest first testing with easy numbers like [1111111111,1111111111] and progressively get more complicated/random. You should do this independently from the 2-2-3-3 shuffle (so set the 3rd input to false). Then to test the 2-2-3-3 shuffle independently, that is like using shunt numbers of all 0's like [0000000000,0000000000] and

then doing the 2-2-3-3 shuffle. That way, you should know exactly what the board should be after it is done.

Another strategy with testing is to just start with a smaller board! That will make tracing/debugging a lot quicker 😊

There are multiple parts of this problem that you will have to figure out how to do algorithmically. Here is a list of some of the things off the top of my head:

- How do you randomly generate a 10-digit number?
- How do you shift a column a specific amount?
- How do you shift a row a specific amount?
- How do interchange between shifting rows and shifting columns? Will you have to write completely different code or could you do something to the array to make things a little easier in between shunting stages?
    - Hint: transpose
- How do you do the 2-2-3-3 shuffle?
- How do you determine if in the new grid a tile is in the same place as the original grid?
    - Hint: masking like oldArr == newArr
- How do you determine if there are alphabetical neighbors?
    - Hint: the diff([‘ABA’]) → [1 -1] (This has 3 alphabetical neighbors).
- How do you determine if there are pairs, triples, fours, or fives next to each other?
    - Hint: the diff(‘AAABBC’) → [0 0 1 0 1] (this has 3 pairs and 1 triple)
- How do you account for if any of the above 2 factors wrap down across a row?
    - Hint: what if you transpose and the linearize to get the array in a vector form
- How do you compute the average metrics over 10,000,000 simulations?

Note: A simulation is defined as a complete execution of shunting and shuffling and then computing the randomness metrics for the new resulting grid of 100 tiles. **Each new simulation should start with the original unshuffled bored. Your function should output the shuffled version of the last board in your simulation, but all numbers outputted should be the averages across all simulations**

Additional note: 10,000,000 simulations is a lot and depending on how you code things it might take a long time. So when coding, you need to think about efficiency. This means only doing steps if you HAVE to, and only doing loops if you HAVE to. If something can be done through just straight indexing, this is usually much faster than looping over all of the elements and doing some complicated stuff. Even if you program takes a long time to do 10,000,000 simulations, maybe just try 1,000,000 or 100,000. How do your metrics compare? Is there a point of saturation where the average numbers you get for let’s say 1,000 simulations are actually very close to 1,000,000?

I will give away up to 50 shirts. The timeline for completing this is until 50 people have done it and could spill into over the summer. You’ll just have to come get them when campus returns to being in-person. In the folder, there will be a link to a spreadsheet that I will update with the names of people who will receive a shirt so you can check that to see how many more people there are until 50 to see if you can still earn a shirt!