Project 1 - Shall we play a game?

In this project, you are asked to implement functions that will enable you (and others) to play the game of sliders.

The objective of this game is to form the board so that all the numbers are ordered on the board. Usually, there is an empty block that will let you move blocks around, but you will use **number 0** for this purpose.

Rules:

- You will implement each of the following functions **with exact names, parameters and default values if given** in your python file.
- Each of your functions should return the expected result in the given form.
- You will be graded for each function.
- Each board structure will be a 2D list (list of lists) in the form of: `[ [1, 2, 3], [4, 5, 0], ... ]` describing a `row x column` matrix.
    - For example, a 2x3 board is passed as `[[0, 1, 2], [3, 4, 5]]`. This represents the actual board as:
      ```
      0  1  2
      3  4  5
      ```
- You should **mutate** the given board. Your grade will depend on this. Do NOT copy or recreate the board unless it is specifically asked.
- The main objective of this game is to try to solve the board by using a predefined set of moves, specifically mutating it to its final form. The final form is the board state after applying the given set of moves.
- For a given board, **the solution** (the solved form) is achieved if all the numbers are in order.
    - For a `3x3` board, the solution is
      ```
      0  1  2
      3  4  5
      6  7  8
      ```
- Board size can go as high as `10x10` and as low as `2x2`, but this should not limit you or change anything in your implementation.

Function Name	**Explanation**
`generate(row, column)`	This function will take two parameters called row and column. Return the generated board. (i.e. the solution) as a list of lists. $10 \geq \{row, column\} > 1$
`shuffle(board, times=20)`	This function will make **times** random shuffles on a given board. The moves will be random, but all the moves will be applied to **number 0.** This will make sure the board is solvable. Make sure that each random move should result in a valid move (meaning it should change the state of the board). Make sure the final board state is not the solution. Note that the given board is mutated at the end of the operation. Return the moves that are made as a string. Use `'L', 'R', 'U', 'D'` for **left**, **right**, **up** and **down** moves respectively. $Times \in \mathbb{Z}^+$
`reset(board)`	This function will reset the board to the original form (solved form). Note that the given board is mutated at the end of the operation. Return None.
`is_valid(board)`	This function will take the board and check to see if the numbers within the board are valid, meaning, if all the numbers are unique and less than `row x column`. Return True if the board is valid, False otherwise. Note that the return type is bool.
`is_solved(board)`	This function will check if the board is solved, meaning if the board is valid and the numbers are ordered (the solution). Return True if the board is solved, False

	otherwise. Note that the return type is bool.
`get_board_size(board)`	This function will take the board and return the size of the board as a tuple. For a `2x3` board it should return `(2, 3)` Return board size as tuple in the form of `(row, column)`
`move_right(board)`	This function will **swap the number 0 with the number to its right** *if it is possible* meaning if there is a number to its right. If the move is not possible, do not make any changes and return the number of moves made which is 0. Note that the given board **is not** mutated at the end of the operation. If the move is possible, make the change, and return the number of moves made which is 1. Note that the given board **is** mutated at the end of the operation. Return the number of moves (Which should be either 0 or 1) as an integer.
`move_left(board)`	This function will **swap the number 0 with the number to its left** *if it is possible* meaning if there is a number to its left. If the move is not possible, do not make any changes and return the number of moves made which is 0. Note that the given board **is not** mutated at the end of the operation. If the move is possible, make the change, and return the number of moves made which is 1. Note that the given board **is** mutated at the end of the operation. Return the number of moves (Which should be either 0 or 1) as an integer.
`move_up(board)`	This function will **swap the number 0 with the number to its up** *if it is possible* meaning if there is a number to its up. If the move is not possible, do not make any changes and return the number of moves made which is 0. Note that the given board **is not** mutated at the end of the operation.

	If the move is possible, make the change, and return the number of moves made which is 1. Note that the given board **is** mutated at the end of the operation. Return the number of moves (Which should be either 0 or 1) as an integer.
`move_down(board)`	This function will **swap the number 0 with the number to its down** *if it is possible* meaning if there is a number to its down. If the move is not possible, do not make any changes and return the number of moves made which is 0. Note that the given board **is not** mutated at the end of the operation. If the move is possible, make the change, and return the number of moves made which is 1. Note that the given board **is** mutated at the end of the operation. Return the number of moves (Which should be either 0 or 1) as an integer.
`move_random(board)`	This function will **swap the number 0 with one of its neighbors randomly.** This should always make a swap, so make sure to pick one direction that has a neighbor. Note that the given board **is** mutated at the end of the operation. Return the move direction (which should be one of `'L'`, `'R'`, `'U'`, or `'D'`) as a string.
`move(board, moves)`	This function will **swap the number 0 with the neighboring numbers according to the given moves string** *if it is possible*. Moves are defined as `'L'` for left, `'R'` for right, `'U'` for up, and `'D'` for down. An example for **moves** list is: `'LRUDDDLUR'`. The inputs will be case insensitive (`'r'` or `'R'` should both be accepted). So `'lrudddlur'` is also a possible input. All other characters should be ignored. (This will not be tested by us, but you should have a proper mechanism to ignore these characters. You should count the number of moves you made. If you cannot make a move *(if the move*

	is not possible), you should not count that to your move number. Return the number of moves as an integer.
`rotate(board)`	This function will rotate the whole board to the right (`90°` rotate), and return the new board. **Note that the board is not mutated, and a new board is returned.** Return the board as a list of lists.
`print_board(board)`	This function will print the given board on the console as a 2D matrix, separate each number in a list with tabs (`\t`), and separate lists with carriage returns (`\n`). Return None
`play(board, moves)`	This function will take a board,and a list of moves, then apply the given moves to the board and check after each move if the board is solved. ● It should first check if the board is valid. If not, return -2. ● If it is solved, it will stop there and return the number of moves it took to solve the board. ● If the board is already solved, it should return 0. ● If the board is not solved at the end of the moves list, return -1. Return number of moves or -1 (if not solved) or -2 (if not valid) as an integer.
`play_interactive(board=None)`	This function will be used to play the game by yourself using manual inputs. (using the **`input()`** function) It should take a valid board as a parameter, and mutate this board. If there is no board entry then this function should ask row and column size as an input. After that function will generate board and shuffle the board (mutated). ● It should first check if the board is valid. If not, return `('', -2)` as a tuple, which is an empty string with -2. ● Next, it should print the board, check if the board is solved and ask for user input. ● If the board is solved in any part, it should print the final board and number of

	moves to the terminal and return the move list and number of moves as a tuple. Example return is `('LLRUD', 5)`. ● With each iteration, it should ask the user for an input {`L, R, U, D, M`} to make moves (left, right, up, down and random), and `Q` to quit the game. The inputs will be case insensitive (`'r'` or `'R'` should both be accepted). Any other letter should be ignored. ● If the board is not solved after the game is quit, it should return the move list and -1 as tuple. Example return is `('LLRUD', -1)`.

An example game

```
>>> b = generate(3, 3)
>>> print_board(b)

0    1    2
3    4    5
6    7    8

>>> is_valid(b)
True

>>> is_solved(b)
True

>>> get_board_size(b)
(3, 3)

>>> m = shuffle(b, times=2)

>>> print_board(b)

1    4    2
3    0    5
6    7    8

>>> print(m)
 'RD'
```

Explanation: 2 moves are randomly chosen. 0 first moved right, then moved down. (This will probably be different in your implementation, and even for each run)

```
>>> reset(b)
>>> print_board(b)

0    1    2
3    4    5
6    7    8

>>> c = move(b, 'RD')
```

Explanation: 2 moves are given. First right, then down. 0 first moved right, then moved down.

```
>>> print(c)
2

>>> print_board(b)

1    4    2
3    0    5
6    7    8

>>> is_valid(b)
True

>>> is_solved(b)
False

>>> d = move_left(b)
>>> print(d)
1
>>> print_board(b)

1    4    2
0    3    5
6    7    8

>>> d = move_left(b)
>>> print(d)
0

>>> print_board(b)

1    4    2
0    3    5
6    7    8
```

```
>>> d = move_up(b)
>>> print(d)
1

>>> print_board(b)

0    4    2
1    3    5
6    7    8

>>> d = move_up(b)
>>> print(d)
0

>>> print_board(b)

0    4    2
1    3    5
6    7    8

>>> v = move_random(b)
>>> print(v)
'D'

>>> print_board(b)

1    4    2
0    3    5
6    7    8
```

Explanation: There were two possible directions for 0 in the board. Down or Right. 0 is randomly moved down.
As an example for the use of random function:

```
list = [1, 2, 3, 4]
print(random.choice(list)) # select random item from list
Output: 3
```

```
>>> f = rotate(b)
```

Explanation: Rotated the board to the right (Columns and rows are swapped).

```
>>> print(f)

6    0    1
7    3    4
8    5    2
```

```
>>> n = play(b, 'LR')

>>> print_board(b)

6    0    1
7    3    4
8    5    2

>>> print(n)

-1

>>> reset(b)
>>> d = move_right(b)
>>> d = move_down(b)
>>> print_board(b)

1    4    2
3    0    5
6    7    8

>>> n = play(b, 'ULRULDUDULR')
```

Explanation: Although more than 10 moves are listed, first two will solve the board and return the solved board with the number.

```
>>> print_board(b)

0    1    2
3    4    5
6    7    8

>>> print(n)

2

>>> reset(b)
>>> d = move_right(b)
>>> d = move_down(b)
>>> print_board(b)

1    4    2
3    0    5
6    7    8
```

```
>>> k,n = play_interactive(b)
Current board is:

1     4     2
3     0     5
6     7     8

Where do you want to move: U

Current board is:

1     0     2
3     4     5
6     7     8

Where do you want to move: Z

Wrong move.

Current board is:

1     0     2
3     4     5
6     7     8

Where do you want to move: L

Current board is:

0     1     2
3     4     5
6     7     8

Congrats! You solved the board in 2 moves.

>>> print_board(b)

0     1     2
3     4     5
6     7     8
>>> print(k)
'UL'

>>> print(n)
2
```

Explanation: Since we called the play_interactive function as k, n = play_interactive(b) it should return the move list and number of moves as a tuple at the end.

```
>>> b = reset(b)
>>> d = move_right(b)
>>> d = move_down(b)
>>> print_board(b)

1    4    2
3    0    5
6    7    8

>>> k, n = play_interactive(b)
Current board is:

1    4    2
3    0    5
6    7    8

Where do you want to move: U

Current board is:

1    0    2
3    4    5
6    7    8

Where do you want to move: Q

Exiting.

>>> print_board(b)

1    0    2
3    4    5
6    7    8

>>> print(k)
'U'
>>> print(n)
-1
```

Explanation: Since the board is not solved and we quit the play_interactive(b) mode using Q, it returns return the move list and -1 as a tuple.

```
>>> k, n = play_interactive()

Please type the puzzle size number.
Row number:3
Column number:3

Current board is:

1     4     2
3     0     5
6     7     8

Where do you want to move: U

Current board is:

1     0     2
3     4     5
6     7     8

Where do you want to move: Q

Exiting.
>>> print_board(b)

1     0     2
3     4     5
6     7     8

>>> print(k)
'U'
>>> print(n)
-1
```

Explanation: Since the board is not solved and we quit the play_interactive(b) mode using Q, it returns return the move list and -1 as a tuple.

```
b = [[1,20,3],[4,15,10]]
>>> k, n = play_interactive(b)
The board matrix elements must be between 0:1:row*column-1
>>> print(k)
''
>>> print(n)
-2
```

Explanation: Since the board is not valid, it returns return the move list as '' and -2 as a tuple.