Function Name: replaceNaN

Inputs:

1. *(double)* An NxM array that may contain some NaN values
2. *(double)* An NxM array that does not contain any NaN values

Outputs:

1. *(double)* An array with doubles in place of the NaNs

Function Description

For some calculations with undefined answers (such as dividing by 0) MATLAB will return the double NaN, for 'not a number'. These values can be problematic if you try to do further calculations on them. For this reason, it may be necessary to remove NaN values from an array before continuing with some computations. Write a function that will take in an array that may or may not contain NaN values and replace the NaNs with the corresponding element of another array that is guaranteed to not contain any NaN values. For example:

```
arr1 = [1, NaN;
        3,   4]
arr2 = [5, 2;
        7, 6]
```

The result would replace the NaN in the first array with 2 from the second array and return the following final array:

```
result = [1, 2;
          3, 4]
```

You **may not** use the `find()` function for this problem.

Notes:

- The two input arrays are guaranteed to have the same dimensions.
- The `isnan()` function will be useful.

Function Name: `camelCase`

Inputs:

1. *(char)* A string of some phrase to be converted into camel case

Outputs:

1. *(char)* The input phrase in camel case

Function Description:

Believe it or not, 'MAT' in 'MATLAB' actually stands for Messaging And Texting*. MATLAB is a tool designed to help you construct that perfect text, especially for Valentine's Day. Write a function called `camelCase()` that will turn an input string into a camel-cased phrase.

Here are the steps to convert a phrase into camel case:

1. Capitalize the first letter of each word, except the first one.
2. Remove all spaces, numbers and extraneous characters.

For example, for the input:

```
'Hey, dinner tonight? :)'
```

The output in camel case would be:

```
'heyDinnerTonight'
```

Notes:

- The input phrase will always begin with a letter, but it may be capitalized or uncapitalized.
- There will always be one space before each word, except for the first word.
- Make sure to use `isequal()` to compare your output with the solution output.
- If you use camel case, people will fall in love with you and your concise and efficient texts.

*This statement not verified by MathWorks.

Function Name: `modIFY`

Inputs:

1. *(char)* A string of any length
2. *(double)* An integer describing the shift

Outputs:

1. *(char)* The encoded message using the designated shift

Function Description:

Have you missed playing around with the `mod()` function? Not to worry! Last week you had so much fun with `caesarSalad()`, but the fun doesn't end there! This week, with the help of masking, you can expand `caesarSalad()` to account for lowercase letters, uppercase letters, and spaces or punctuation. If you thought it couldn't get any better, the function name is also now a pun! You're welcome.

Given a string of any length and an integer containing a shift number, shift every **letter** the appropriate amount to produce an encoded message. Do not shift punctuation or spaces. Capital letters should stay capital and lowercase letters should stay lowercase. For example, a `'Z'` shifted by 1 would wrap around to `'A'`, whereas a `'z'` would wrap to `'a'`.

Notes:

- Your code should work for positive shifts as well as negative shifts, and the value of the shifts have no limit.

Hints:

- The `mod()` function can be used with both positive and negative numbers.
- Consider shifting the uppercase and lowercase letters separately and using `caesarSalad()` as a helper function.

Function Name: `checkMagic`

Inputs:

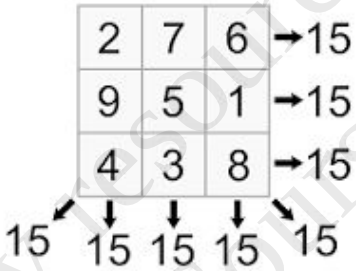
1. (*double*) An array of integers

Outputs:

1. (*logical*) A logical representing whether the array is a magic square

Function Description:

As we are exploring the wonders of MATLAB, we find cooler and cooler things that it can do. One of the best things about MATLAB is that it can do magic! Well...not really...but it can produce a magic square on command with the `magic()` function, which is pretty close. If you don't remember from your 4th grade math class, a magic square is a matrix of numbers where the rows, columns, and diagonals all add up to the same number. For example, the following is a valid magic square:



Write a function in MATLAB that takes in an array and checks whether the array is a magic square by outputting a logical. Remember the key aspects to check for a magic square: the array must be square, all elements must be unique, all columns and rows add to the same number, and *both* diagonals add to the same number as the rows and columns.

Notes:

- You **may not** use the `diag()`, `unique()`, `trace()`, or `eye()` functions.
- A 1x1 matrix is a magic square.
- All numbers will be integers.

Hints:

- Think about how you can use linear indexing to get the diagonal of the matrix without using any of the banned functions.

Function Name: `puzzleBox`

Inputs:

1. (*char*) A jumbled character array
2. (*double*) The row shift vector
3. (*double*) The column shift vector

Outputs:

1. (*char*) The solved character array

Function Description:

Have you ever had the urge to get a wooden Chinese Puzzle Box and spend your free time trying to solve it, but realize that you have no free time? Well have no fear, now you'll be forced to solve one! Write a function called `puzzleBox` that will take in a jumbled character array and two vectors (representing a row shift and a column shift) and produce a satisfying ASCII image. The last elements in the row and column shift vectors represent the number of shifts to make, while all other numbers in the vector represent which row/columns to shift. For example, if the row shift vector was `[3, 5, 12, 7]`, this would indicate that you should shift every element in row 3, 5 and 12 to the right by 7 columns (shifts should wrap around to column 1 if they go past the right edge of the array). Positive row shifts move elements to the right and positive column shifts move elements down. Negative shifts move in the opposite direction. Your code should be formatted such that the rows slide first, and then the columns.
(example on next page)

For example, given the following inputs:

```
jumbledCharArr = ['abc';...
                  'def';...
                  'ghi']

rowShift = [1, 3, 2];

colShift = [2, -5];
```

You should first shift rows 1 and 3 by 2 elements to the right (wrapping elements around the end of each column). This procedure will produce the array

```
afterRowShift = ['bca';...
                 'def';...
                 'hig']
```

Then you should shift column 2 by 5 elements up (wrapping elements around the end of each row). Because the array is a 3x3, shifting by 5 in the vertical direction is equivalent to shifting by 1 in the downward direction (**cough** `mod()` **cough**). Therefore, the final character array produced by these inputs is:

```
finalCharArr = ['bia';...
                'dcf';...
                'heg']
```

Notes:

- The shift value can be any integer.
- The index elements will always consist of valid row/column indices

Hints:

- You will find the `mod()` function very useful.
- Character arrays are just normal arrays in disguise.