Homework #04: Logicals & Logical Indexing

Drill Problem #1

Function Name: meanEven

Inputs:

1. (*double*) A vector of random, integer numbers

Outputs:

1. (*double*) The average of the specified type of numbers

Function Description:

In this problem, you are given a vector of random numbers. Using logical indexing, isolate only the even numbers from the vector, and then output their average.

Notes:

– For the sake of this function, treat zero as an even number.

Hints:

– The `mod()` function will prove useful in this function.

Drill Problem #2

Function Name: twinPrimes

Inputs:

1. (*double*) A vector of random integers

Outputs:

1. (*logical*) A vector describing whether each number of the original input vector is a member of a pair of twin primes

Function Description:

Twin primes are a phenomenon in mathematics where a pair of prime numbers exist and differ by only two. For example, 17 and 19 are both twin primes because they are prime numbers and the difference between them is 2. Another example of twin primes are the numbers 41 and 43. It is conjectured that an infinite number of twin primes exist, and proving this remains one of the most elusive problems in number theory to this day (Wolfram MathWorld).

For more information on twin primes, you may visit:

http://mathworld.wolfram.com/TwinPrimes.html

For example, if an input vector contained the following numbers:

```
vec = [41 23 13 4 11 43 13 8]
```

Then your vector plus or minus two will be the following:

```
vecPlus2 = [43 25 15 6 13 45 15 10]     vecMinus2 = [39 21 11 2 9 41 11 6]
```

The values in the first index of both `vec` and `vecPlus2` are prime numbers, so the first number in `vec` is a member of a twin prime pair. The values in the third index of `vec` and `vecMinus2` are also both prime, so the third number in `vec` is a member of a twin prime pair. The output of this function with `vec` as the input will be the following:

```
out => [true false true false true true true false]
```

Given an input vector of random integers, write a function in MATLAB that will determine which of the numbers in the input vector are members of a pair of twin primes.

Notes:

- You must determine if the numbers in the vector are members of a pair of twin primes, but each one could be the first in the pair or the second in the pair.
- Each number in the vector will be of the value three or greater. This should not affect your code: it is to account for any corner cases (that could have potentially messed you up) involving negative numbers and the `isprime()` function.

Hints:

- The `isprime()` function may come in very handy for this function.

Drill Problem #3

Function Name: rhymeThyme

Inputs:

1. (*char*) A string containing two sentences separated by a single period

Outputs:

1. (*logical*) A logical (`true` or `false`) value specifying if the lines provided rhyme

Function Description:

Now that you've finished the first few weeks here at Georgia Tech, you are missing your appreciation for liberal arts and all its wonders! Shall I compare thee to a summer's day? Sure, Shakespeare, why not! And "where for" art thou Romeo? He sits behind you in Physics, duh!

Write a MATLAB function `rhymeThyme` to help you evaluate some lines of poetry, based on your knowledge of strings and logicals. For this problem, two sentences are said to rhyme if the following two conditions are met:

1. The last 3 characters of each sentence are identical (not including the final period).
2. Each sentence has the exact same number of words.

Write a function that checks a single string input containing two sentences for these two conditions; the function should return a logical `true` if both conditions are both met and a logical `false` if they are not.

For example, if the input string were the following:

```
str = 'I can rhyme. All the thyme.'
```

Then the output would be `true` because the last three characters of each sentence are the same (`'yme'`) and both sentences have 3 words in them.

Notes:

- The input string will contain two sentences which are always separated by a single period in between.
- Each sentence in the single string is guaranteed to have at least 3 characters in it.
- The different words in the input string are guaranteed to be separated by a single space. You can ignore all extraneous punctuation when considering the number of words in each sentence.
- You should not actually use this function for practical applications to check if words rhyme. If you use this to check your poetry for a literature assignment, it may let you down.

Drill Problem #4

Function Name: vecResize

Inputs:
1. (*double*) A vector of length N
2. (*double*) A scalar value by which the vector should be stretched or compressed

Outputs:
1. (*double*) The new stretched/compressed vector

Function Description:

Write a function in MATLAB called `vecResize()` that inputs a vector with at least one element and a value by which the vector should be stretched or compressed. This function will output a new vector that has been resized based on the scaling factor given in the second input. If the scaling factor is greater than 1, then the vector should be stretched by adding duplicate elements. If the value is less than 1, then the vector should be compressed by removing elements. If the value is exactly 1, then no change should be made to the vector.

For example, if the following values were input:

```
vec = [0, 1, 1, 2, 3, 5, 8, 13]
scalingFactor = 2
```

then the resulting vector would be stretched by a factor of two by adding duplicate elements:

```
newVec => [0, 0, 1, 1, 1, 1, 2, 2, 3, 3, 5, 5, 8, 8, 13, 13]
```

However, if `scalingFactor = 1/2`, then the vector would be compressed by a factor of two by removing elements from the vector:

```
newVec => [0, 1, 5, 13]
```

Your function should be able to account for any positive, non-zero scaling factor.

Notes:
- The input vector is guaranteed to contain at least one element.
- The `linspace()` function, in conjunction with the `round()` function and numerical indexing, are useful in solving this problem.

Drill Problem #5

Function Name: criminalMinds

Inputs:

1. (*logical*) Vector of suspect #1's answers to a lie detector
2. (*logical*) Vector of suspect #2's answers to a lie detector
3. (*logical*) Vector of suspect #3's answers to a lie detector
4. (*logical*) Vector of suspect #4's answers to a lie detector

Outputs:

1. (*char*) Sentence stating which suspect is lying

Function Description:

After all of those years of reading Nancy Drew and watching Bones, you realize that your true passion lies in justice and bringing criminals to light. After tedious years of training with the FBI, you are finally working a case, and you have four suspects—only one of which is the true criminal. You give them each a separate lie detector test and plan to use the results to find which of the four suspects is lying to you. Each suspect who is telling the truth will give the same corresponding yes or no (`true` or `false`) answers, since that is how real life innocent suspects behave, while the suspect who is lying will not corroborate his/her answers with the other three. Since you were a pro at MATLAB back in your engineering days, you decide to write code to assist you in finding the criminal.

Write a function with four inputs, where each input is a logical vector corresponding to the answers a suspect gives on the lie detector. Three of the suspects will have the exact same answers, but one suspect's answers will be slightly—or completely—different than the others' answers. Using your knowledge of logical indexing and masking, find which of the four suspects is lying, and, thus, is the criminal. The output string will be of the form `'Suspect #_ is lying.'` Where the `'_'` corresponds to suspect number who is lying and the number is from the input order.

Notes:

- The suspect who is the liar will have *at least one* answer that is different from the other suspects' answers, but could differ up to every answer.
- You may not use the `isequal()` function to code this problem. Use of the `isequal()` function will result in zero credit for this problem.

Drill Problem #6

Function Name: caesarCipher

Inputs:

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

Outputs:

1. (*char*) A coded string using the Caesar cipher

Function Description:

Last week you wrote the function `caesarSalad()` to apply a Caesar shift to lowercase letters and encode a word. Now, we will extend that cipher to entire phrases! Using the base code from last week as a skeleton for one (or two) helper function(s), write a function in MATLAB named `caesarCipher()`. This function will take in a string with upper and lower case letters, as well as punctuation, and then apply the Caesar shift to the string, using the helper function(s) you have created to encode it. Only letters (both upper and lower case) should be encoded using the Caesar cipher. Any other characters, such as spaces, periods, etc., should not be altered.

Notes:

- The cipher should work for both positive and negative integers that indicate the shift as given by the second input.
- There is no limit to the value of the shift number in the second input.
- You MUST use a helper function to solve this problem.
- You may write your helper function(s) below your `caesarCipher()` function, or you may write it/them to separate files. *If you write it/them to a separate file(s), you MUST include it/them in your overall file submission.*

Hints:

- The `mod()` function, and its effect on both positive and negative numbers, may be useful.
- Consider editing `caesarSalad()` to have one helper function for capital letters and one for lowercase letters.