Function Name: `fivePrime`

Inputs:

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

Outputs:

1. (*double*) The original vector with `NaN`s in it

Function Description:

You are given a vector of random, positive integers. Using logical indexing, find the numbers divisible by 5 and all the prime numbers in the vector, and calculate their average (rounded to the closest integer). If that average occurs anywhere in the original vector, replace the number with an `NaN` and output the new vector.

Notes:

- You should use `isequaln()` to check your answer with the solution.
- You will never be given negative integers.

Hints:

- The `isprime()` function may prove useful.

Function Name: `giraffeCase`

Inputs:

1. *(char)* A sentence to be converted to giraffe case

Outputs:

1. *(char)* The sentence converted to giraffe case

Banned Functions:

`strrep()`

Function Description:

You might have heard of camel case, the naming convention typically used when naming variables and functions in MATLAB, but have you ever heard of giraffe case? Probably not, since we just made it up!

Write a function that converts a given phrase to giraffe case according to the following rules:

1. Capitalize all letters that occur at the end of a word, and make all other letters lowercase.
2. If `'giraffe'` appears in the string with any capitalization, capitalize all its letters (`'GIRAFFE'`).
3. Remove all spaces and non-letter characters.

For instance, the string `'Giraffes are just long horses!'` would be converted to `'GIRAFFESarEjusTlonGhorseS'`.

Notes:

- Punctuation may occur anywhere in the string.
- `'giraffe'` will not appear in a string more than once.
- `'giraffe'` may be a substring of another word so if `'giraffezone'` occurs in the string, it should become `'GIRAFFEzonE'`.
- Each word will be separated by at least one space.
- The string will always begin with a letter.

Function Name: `caesarShift`

Inputs:

1. *(char)* A string to be encoded
2. *(double)* The shift number

Outputs:

1. *(char)* An encoded string

Banned Functions:

`strrep()`

Function Description:

The Caesar shift was a method that Julius Caesar, a famous Roman general, used to encode his messages. Caesar shifted each letter over 3 places, so that `'a'` became `'d'`. The shift wraps around the end of the alphabet, so the letter `'z'` would shift to become the letter `'c'`. However, the Caesar shift is relatively easy to crack today, so in order to encrypt your dank memes, you decide to make it harder to crack by adding some more steps. Being a MATLAB pro, you decide to write a function that does it for you!

The function takes in a string to be encoded and a shift number representing how far each letter will be shifted for the Caesar shifts you will perform. It then performs the following actions in order on the input string:

1. Convert all letters to uppercase, since classical Latin had one case.
2. For letters with even ASCII values, perform a Caesar shift using the given shift number.
3. For letters with odd ASCII values, perform a Caesar shift using the negative of the shift number.
4. Replace all instances of `'J'` with `'I'` and `'U'` with `'V'`, since classical Latin had no J's or U's (Julius Caesar was written as IVLIVS CAESAR).
5. Concatenate the number of consonants in the resulting string with the output string.

Notes:

- The function should work for both positive and negative shift values of any magnitude.
- Consonants are defined as letters that are not A, E, I, O, or U.
- The input string is guaranteed to only consist of letters and spaces. Classical Latin did not use punctuation.

Hints:

- The `mod()` and `num2str()` functions will be useful.

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

Banned Functions:

`isequal()`, `isequaln()`

Function Description:

After years of reading Nancy Drew and watching Bones, you realize your true passion lies in criminal justice. After years of training with the FBI, you are finally working a case, and you have four suspects—only one of whom is the criminal. You give each one a lie detector test and use the results to find which of the four suspects is lying. Each suspect who is telling the truth will give a corresponding yes or no (`true` or `false`) answer to each question, while the suspect who is lying will not corroborate at least one of his/her answers with the other three. Since you were a pro at MATLAB back in the day, you decide to write code to assist you in finding the criminal.

Each input to the function is a logical vector corresponding to the answers a suspect gives on the lie detector. Each element of the vectors represent a different question. 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, determine which of the four suspects is lying, and thus, is the criminal.

The output string will be of the form `'Suspect #<num> is lying.'`, where `<num>` corresponds to the suspect number who is lying. The number is determined by 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. However, the use of `isequal()` to check that your output matches the solution outputs is encouraged.
- The output string ends with a period. Do not forget that period!