For this homework, many of the outputs are of class logical. Remember that MATLAB displays logical values as `0` or `1`, not `true` or `false`. However, these are not doubles and you will get the problem wrong if your function outputs the double `1` instead of the logical `1` (`true`). For this reason, be sure to check both the value and the class of your outputs with the solution function.

Happy coding!
~Homework Team

Function Name: `grammarCheck`

Inputs:

1. *(char)* A simple sentence

Outputs:

1. *(logical)* A logical value indicating whether the sentence is grammatically correct

Function Description:

As a hardworking Tech student, you decide to use MATLAB to ensure that you are using proper grammar in all of your lab reports and essays. Write a function that takes in a simple sentence and outputs a logical value indicating whether the sentence is grammatically correct.

In order to have correct grammar, your sentence must meet the following three criteria:

- The sentence must start with a capital letter
- The sentence must end with either a period, a question mark, or an exclamation point
- Since it is Tech tradition to abbreviate the University of Georgia as u[sic]GA, the string `'UGA'` (case sensitive) cannot appear anywhere in the sentence

Notes:

- The sentence is guaranteed to be non-empty.

Function Name: `comboLock`

Inputs:

1. *(char)* A string indicating the letter and number combination applied to the lock
2. *(double)* Minimum value of numeric digits
3. *(double)* Maximum value of numeric digits

Outputs:

1. *(logical)* Whether or not the input combination meets the requirements

Function Description:

Cyber security is more important than ever, as more and more of our sensitive data is stored online. Combinations of character and numeric values must be more and more complex and meet certain minimum security requirements. Write a MATLAB function, `comboLock`, which takes in a string containing a series of letters directly followed by a series of numbers and determine if it meets the specified combination rules. The output should be a logical `true` or `false` (indicating if the combination met the specified requirements).

Combination Requirements:

- There must be more letters than numbers in combination.
- There must be at least one uppercase letter.
- The summation of all numeric digits must be *between* the given minimal and maximal values (exclusive).

Notes:

- The input combination string will always have only upper and lower-case letters directly followed by a series of only numeric digits.
- Be sure to check the data type of your output!

Function Name: `gibberish`

Inputs:

1. *(char)* A sentence written in an unknown language
2. *(char)* A dictionary of words, separated by spaces
3. *(char)* A noun word list, separated by spaces
4. *(char)* A verb word list, separated by spaces
5. *(char)* An adjective word list, separated by spaces

Outputs:

1. *(logical)* A logical `true` or `false` for whether the sentence is valid.

Function Description:

In today's ever expanding world and economy, and now with even the possibility of extraterrestrial life, there is a growing need to check the legitimacy of a language. Use MATLAB and the rules below to check whether a sentence written in an unknown language is valid. The following are the rules that define a valid language:

1. All words in the sentence must be present in the dictionary.
2. Sentences are arranged in one of the following orders:

Noun → Verb → Noun/Adjective

(If first word is a noun, second should be a verb, and third should be a noun or adjective)

Adjective/Verb → Noun → Verb/Noun

(If first word is an adjective or verb, second should be noun, and third should be verb or noun)

3. `grammarCheck_soln()` should be valid for the sentence.

You should output a logical true or false depending on whether the sentence is valid under these rules. The dictionary and noun/verb/adjective lists are stored in the .mat file 'dictionary.mat'. The sentences are stored in 'sentences.mat'. You can load the test cases using the `load()` function.

Notes:

- Words will always separated by spaces.
- You are guaranteed to have three and only three words in the sentence.
- The sentence will always end with some sort of punctuation.
- You **must** call the **`grammarCheck_soln`** function for the 3rd rule; **do not** call your `grammarCheck` function.
- **Do not** submit `grammarCheck_soln` to T-square.
- Matching words with the various word lists should ignore case.
- Any given word will only be found in at most one of the part-of-speech lists.
- Any word contained in the dictionary is guaranteed to occur in one of the other lists.

Hints:

- Make some small test cases for easy debugging!

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()`

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 polygraph 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, find 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 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.