Notice

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: `passwordProtec`

Inputs:

1. *(char)* A 1xN string of a user's password

Outputs:

1. *(logical)* A `true` or `false` depending on whether the password passes the requirements

Function Description:

In today's society, hackers are the new villains, and cyberspace is the new battlefield. Some hack for good, others for bad - they protec, they attac - but most importantly, Tech's on track. To prevent you from becoming the latest victim of cybercrime, Georgia Tech is rolling out two-factor authentication, with mandatory enrollment by October 16th. The good news is, your password will no longer require 47 characters, a pictogram, the first 19 digits of pi, and your first-born child, nor will it need to be changed every three months. But what should you choose for your new, simple password? Will it still meet the basic requirements? Use MATLAB to create a function checking whether or not a certain password passes the required criteria. It should output `true` if it meets the requirements, `false` if it does not.

Criteria:

1. Must contain at least 8 characters.
2. Must contain at least one lowercase letter, one uppercase letter, one number, and one special character.

Notes:

- For this problem, a special character is any character that is not a letter or number.
- These are not the official requirements for a password.

Function Name: `suspects`

Inputs:

1. (*char*) A 1x(3*N) vector of initials
2. (*logical*) A 1xN vector of whether or not each suspect looks familiar

Outputs:

1. *(char*) A 1xP vector of the initials of the people mistaken for suspects
2. (*double)* A 1xM vector of the original positions of the remaining suspects

Function Description:

While working on your CS 1371 homework, a villain rushes in and steals your laptop with all of your MATLAB code on it! You think you saw the thief's face, so you file a case with the GTPD. They then gather a group of suspects who meet the description you provided for the Clery Act, and line them up for you to identify. You decide to code a MATLAB function (on the mobile app, since your laptop was stolen) to help you quickly identify who could have stolen your laptop.

Given a list of the initials of the suspects in the line-up and a logical vector of whether you recognize each suspect, output the initials of those who were wrongly accused - that is, those you do NOT recognize. The initials should be output in alphabetical order, based on first initial. You should also output a vector of the positions of the suspects you recognize in the line-up, in ascending order.

Example:

The following gives an example of the two outputs for the following test case:

```
initials =  'AV AL WL NB EB AS '
familiar =  [true true false true true false]

mistaken = 'AS WL '
positions = [1 2 4 5]
```

Notes:

- Initials will always be two capital letters followed by a space, both in the input and output.
- In the case of a tie in the first initial, the order should be based on the original position in the lineup, not the second initial.

Hints:

- Applying the logic from a previous homework problem may be helpful.

Function Name: `lockSMITH`

Inputs:

1. *(double)* A 1xN vector of potential numbers
2. *(double)* A 1x5 vector of rotations
3. *(logical)* A 1x5 vector of logicals

Outputs:

1. *(double)* A 1xM vector locker combination

Function Description:

You have locked yourself out of your room and your roommate is too busy working out at the CRC to come let you in. Since your roommate already took CS 1371, she decides to test your skillz by sending you three vectors along with instructions on how to open her CRC locker so that you can borrow her key.

The first input contains the potential numbers from which you will pull out the correct combination. The second vector represents the number of positions you have to "rotate" to the left (negative values) or to the right (positive values) to get the combination from the first vector. You should always start your first "rotation" in between the first and last elements of the vector, so a rotation of `1` would mean the first element should be saved and `-1` would mean the last element. Further rotations should begin at the number that was last saved. The third and final vector is used to determine the "true" combination, where logical trues correspond to numbers you should keep in your final output combination. The order of numbers in the final combination should match the order in which they were pulled out from the first vector.

Example:

Potential (in1)	`[48 31 22 21 41 36 12 83 92 10]`
Rotations (in2)	`[4 -2 6 1 2]`
Confirmed	`[21 31 83 92 48]`
Logical (in3)	`[true false false true true]`
Combination (out1)	`[21 92 48]`

Notes:

- If the number of rotations exceeds the length of the vector of potential numbers in any direction, the vector of potential numbers should be wrapped around so that you return to the first element after reaching the end and vice versa.

Function Name: `superheroCrisis`

Inputs:
1. (*double*) A 1x5 vector of the initial villain counts in each city
2. (*double*) A 1x5 vector of the number of villains in the first wave that attacks each city
3. *(double)* A 1xN vector of the number of villains in the second wave that attacks each city

Outputs:
1. (*char*) A single character giving the initial of the last city standing
2. (*logical*) A `true` or `false` based on whether the heroes successfully defeated Thanos

Function Description:

After years of biding his time, Thanos is ready to make his move and attack key world cities. These are the limits of how many villains each city's heroes can defend against at once:

New York City `'N'` Spider-Man Marvel's Defender	**Asgard** `'A'` Thor Loki	**Wakanda** `'W'` Black Panther Wolverine	**Los Angeles** `'L'` Iron Man War Machine	**Washington D.C** `'D'` Black Widow Captain America
25	40	15	35	50

Thanos plans to attack in waves, sending an initial army to each city and then later sending reinforcements. A city will survive the first wave if the total number of villains originally in the city (the first input) and in the first wave of reinforcements (the second input) does not exceed the capacity listed in the table above. If a city survives the first wave, half of the total villains in that city will be defeated. Of the remaining cities, each will survive the second wave if the total number of villains remaining in the city from the first wave and in the second wave of reinforcements (the third input) does not exceed the capacity listed in the table above. Only the one city defended by the fiercest heroes will survive this second wave, defeating one third of the villains present in the city at the beginning of the second wave. Output the initial of this valiant city. After the second wave of attacks, Thanos decides it is time for him to enter the battle against Marvel's heroes and wipe this final worthless city off the map himself. If, when Thanos attacks, the number of villains remaining in the last city is greater than half the limit of that city, listed in the table above, Thanos will defeat the heroes and win the battle. Output a logical `true` if the heroes are successful in defending against Thanos or a `false` if the heroes perished alone in the streets trying to save Earth from the tyrannical clutches of Thanos.

Notes:
- The first two inputs always correspond to this order of city initials: `'NAWLD'.`
- The third input is also in this order, but includes only the surviving cities.
- Round values to the nearest whole number after each of the first two waves.
- If a city is destroyed, the villains in that city no longer pertain to the problem.

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!