Function Name: `scales`

Inputs:

1. *(double)* Vector of numbers representing notes
2. *(char)* Note you want to count

Outputs:

1. *(char)* Vector of note names
2. *(double)* Number of times the note specified in the input occurs

Background:

You are an artist about to perform at Music Midtown. Before you go on stage, you better warm up! Normally you warm up by singing musical scales, but you decide try something different and use the set of warmups your friend gave you beforehand. However, she gave them to you in a weird format. You decide to use MATLAB to convert your friend's warmups into the notes that you understand!

Function Description:

You have been given a vector of integers greater than or equal to 0. You want to make sure that you don't strain your voice before the big show, so you limit the range by deleting every number representing a note greater than 18. Each number corresponds to one of seven musical notes, whose names range from 'A' to 'G'. The numbers {0, 7, 14, ... } correspond to the note 'C', the numbers {1, 8, 15, ... } correspond to 'D', etc. The numbers {5, 12, 19, ... } correspond to 'A' and {6, 13, 20, ... } correspond to 'B'. Output a character vector of the note names that correspond to the numbers in the first input. Then, count how many times the note specified in the second input occurs in the vector, and output that count as the second output.

Example:

```
[notes,count] = scales([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14], 'C')

      notes = 'CDEFGABCDEFGABC'
      count = 3
```

Notes:

- The `mod()` function will be useful.

Hints:

- Turn on your volume and run the solution function a bunch for some cool easter eggs!

Function Name: `lyrics`

Inputs:

1. (*char*) A character vector containing the title and artist of a song, separated by a comma
2. (*char*) A character vector containing the lyrics to a song

Outputs:

1. *(char*) A vector of characters describing your opinion of the song

Background:

All your life you have aspired to become the greatest DJ in the world. Soon, you will have the chance to prove yourself- as the headlining DJ of Music Midtown! Not wanting to ruin this once in a lifetime opportunity, you want to ensure all the songs you remix and sample are the best of the best. You realize there's not enough time to go through thousands of lyrics, so you use your best friend MATLAB to help you get the job done!

Function Description:

Write a function that uses the following criteria to rate a song's lyrics:

1. If the song mentions either `'MATLAB'` or `'1371'` at least once, it earns 20 points.
2. If the song is performed by Taylor Swift, it earns 10 points.
3. If the song is performed by Kanye West, it loses 5 points.
4. If the song mentions your favorite color, `'gold'`, and does not mention your least favorite color, `'red'`, it earns 15 points.

Your function should output a vector of characters in the following format:
`'<song title> received a score of <score> points.'`

Example:

```
>> out = lyrics('Delicate,Taylor Swift', 'This ain't for the best. My reputation's never been gold, so you must like me for me. Yeah, I want you. We can't make 1371 promises now, can we, babe?')

out => 'Delicate received a score of 45 points.'
```

Notes:

- `'Taylor Swift'`, `'Kanye West'`, and `'MATLAB'` will appear exactly as shown, case sensitive.
- You should ignore case when finding `'red'` and `'gold'`. If these words appear as part of another word, you should still count them.
- You should not use conditionals to solve this problem.

Continued...

Hints:

- Performing a mathematical operation on a logical will result in its automatic conversion to a double.
- Vectors of the same length can be multiplied together elementwise using `.*`
- You may find the `strtok()` and `sprintf()` functions to be useful.

Function Name: `trackList`

Inputs:

1. (*char*) A 1x(6*N) vector of song names
2. (*logical*) A 1xN vector of whether or not each song is on that artist's tracklist

Outputs:

1. *(char*) A 1xP vector of the song names that should not be on the tracklist
2. (*double*) A 1xM vector of the original positions of the actual songs

Background:

Music Midtown 2018 is coming up and you're really excited to go! The lineup of all the artists performing was just released and to prepare for the festival you're going to go through and listen to some of the artists' music. But Spotify just got hacked and now the tracklists of different albums have the wrong songs in them! You decide to use MATLAB to figure out which songs in the tracklist were actually sung by the artist.

Function Description:

Given a list of the first 5 letters of the songs in a mixed up tracklist and a logical vector of whether the song belongs in the tracklist, output the songs that should not be included in the tracklist. You should also output a vector of the positions of the songs that belong in ascending order.

Example:

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

```
>> songs = 'FLASH HOMEC MASKO DESPA CANTT STRON '
>> belong = [true true false false true true]
>> [wrong, positions] = trackList(songs, belong)
            wrong => 'MASKO DESPA '
            positions => [1 2 5 6]
```

Notes:

- Song names will always be five capital letters followed by a space, both in the input and output.
- You will not be given a test case with no falses in the logical input vector.

Function Name: `caesarShift`

Inputs:

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

Outputs:

1. *(char)* An encoded string

Function Background:

You're going to Music Midtown '18! 18 BCE that is, of course! You're a famous artist in Roman times. You want to send the lyrics to be edited, but you don't want the couriers to read it, so you decide to encrypt your lyrics. You remember that your friend, Julius Caesar, had a great method for encoding 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 all of the lyrics, you decide to make it harder to crack by adding some more steps. Being a Roman MATLAB pro, you decide to write a function that does it for you!

Function Description:

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, including the `'V'`s added in Step 4.

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 (and not spaces).
- The input string is guaranteed to only consist of letters and spaces. Classical Latin did not use punctuation.

Hints:

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

Function Name: `favoriteBand`

Inputs:

1. (*double*) An array of numbers representing the schedule of performances for the day
2. (*double*) A 1xM vector of when you are expecting rain

Outputs:

1. *(char*) A character vector describing when and where your favorite band is playing
2. (*double)* An array of numbers representing the updated schedule

Background:

You've finally made it to Music Midtown, and you are absolutely overwhelmed by the vast number of amazing performances that will be going on all day. How are you going to be able to decide which band is your top priority?? Oh we know, MATLAB!

Function Description:

Based on the schedule input and the expected rain input, you will figure out when and where your favorite band is playing after taking the rain into account. In the first input, the rows represent the order of bands playing (ex: bands playing second are in row 2) and the columns represent the different stages (ex: bands playing on the first stage are in column 1). First, account for the rain by deleting the row (or rows) that corresponds to the act (or acts) during which it will rain. Then, find the stage and order that your favorite band is playing. Your favorite band will have the maximum value in the array. Finally, output a character vector describing when and where they are playing in the following format:

```
'My favorite band is act <num> playing on Stage <stage number>!'
```

Example:

```
>> schedule = [3   7   2; 6   1   9; 11  2   8; 8   5   3]
>> rainTime = 3
>> [fave, sched] = favoriteBand(schedule, rainTime)
fave => 'My favorite band is act 2 playing on Stage 3!'
sched => [3  7  2; 6  1  9; 8  5  3]
```

Notes:

- There is guaranteed to only be one top band; the greatest number in the array will be unique.
- The second input can be either a scalar or a vector.

Hints:

- You may find the second output of `max()` useful.
- Use %d with `sprintf()` for the stage and order numbers.

Extra Credit

Function Name: `setThief`

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 stole the set list

Banned Functions:

`isequal(), isequaln(), strcmp(), strcmpi()`

Background:

You are the manager for the headliner at a music festival, and there's been an incident: someone stole and leaked the set list. The band is not happy, and it's up to you to bring justice to the thief. You have identified four suspects, and given them a lie detector test. You know that each suspect who is telling the truth will have matching answers. However, the thief will having at least one answer that is different from the rest. You have decided to use your new MATLAB skills to find the true thief.

Function Description:

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 set list thief.

The output string will be of the form `'Suspect #<num> leaked the set list.'`, where `<num>` corresponds to the suspect number who stole the set list. The number is determined by the input order.

Notes:

- The suspect who lied (the one who leaked the set list), 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!