Function Name: `atbash`

Inputs:

1. *(char)* A single, all lower-case word

Outputs:

1. *(char)* The word with a modified Atbash cipher applied

Function Description:

The Atbash cipher is a simple substitution cipher. It involves replacing each letter of a string with the "opposite" letter of the alphabet. 'a' is replaced with 'z', 'b' is replaced with 'y', and so on. For this function, once you switch each letter with it's opposite, also reverse the string. For example, if the input is `'hello'`, applying the Atbash cipher produces `'svool'`. Then reversing the string produces `'loovs'`.

The input string is guaranteed to only contain lowercase alphabetic characters (no numbers, spaces or special characters).

One interesting property of the Atbash cipher is that the exact same function can be used to encode and decode words. So `atbash(atbash('hello'))` simply returns `'hello'`.

Notes:

- Use the `char()` function to convert a vector of ASCII values into their character equivalent.

Hints:

- Remember that strings are really just vectors of ASCII values and can be manipulated the same way that numerical vectors can be manipulated.

Function Name: `potatoGrowth`

Inputs:

1. *(double)* A 1xN vector of potato plant heights in inches
2. *(double)* A 1xN vector of days

Outputs:

1. *(char)* A string describing the highest range of growth

Function Description:

Suppose there is an astronaut that gets stuck on Mars after being injured in a wind storm that forces the rest of the crew to abandon their mission. To survive, this astronaut must grow potatoes. The astronaut documents the height of the plants each day and wants to use MATLAB to calculate when the plants grew the most. This function will take in a vector of heights and a vector of days. These two vectors will always be the same length. Find the two days between which the potato plant grew the most and output these two days in a string formatted as follows:

`'The potato grew <num> inches between day <day1> and day <day2>'`

`<num>` should be replaced with how much the potato grew between the two days. `<day1>` should be the start of the day range and `<day2>` should be the end of the day range.

The days are not necessarily consecutive. For example, if the heights vector is `[1, 7, 2, 3]` and the days vector is `[1, 10, 4, 5]` then the plant grew the most between day 5 and day 10 (4 inches). Therefore, the output would be:

`'The potato grew 4 inches between day 5 and day 10'`

Notes:

- A potato plant can wilt and die, so there is no guarantee that the plant always gets taller as time goes on.
- There will never be a tie for the greatest growth.
- There will be at least one range of days for which the plant's height increases.

Hints:

- You may want to utilize the second output of `sort()` and `max()`.

- Also check out the `diff()` function.

Function Name: `packNums`

Inputs:

1. *(double)* A 1xN vector of positive 3-digit integers

Outputs:

1. *(double)* A 1x(3*N) vector of all of the digits from the original vector as separate integers

Function Description:

Take a vector of positive 3-digit integers and split the values into a vector of single integers, where each number appears in the same order as the original vector. For example, if the input is `[425, 712, 804]` the the output should be `[4, 2, 5, 7, 1, 2, 8, 0, 4]`. All of the numbers appear in the same order, they are just split into single integers and 'packed' back into a vector.

You are encouraged to use and/or modify your `sepDigits()` function from last week to help you with this function.

Function Name: `stretchVec`

Inputs:

1. *(double)* A 1xN vector
2. *(double)* A positive, integer scale value

Outputs:

1. *(double)* A 1x(scale*N) vector with each element repeated 'scale value' number of times

Function Description:

Write a function that duplicates the values of a vector by some scale factor. The values should maintain their respective order. For example, if the vector was `[5, 12, 4, 4, 8]` and the scale factor was 3, then the result should be
`[5, 5, 5, 12, 12, 12, 4, 4, 4, 4, 4, 4, 8, 8, 8]`. Each element of the original vector is repeated 3 times.

Notes:

- You MAY NOT use the `kron()` function.
- Zero is not a positive number.

Hints:

- Play around with the `linspace()` and the `floor()` function. Think about how these function could be used to produce the indices that you need.
- Using `linspace(1, length(vec), length(vec)*scale)` will get you moving in the right direction. See if you can figure out where to add `(scale - 1)/scale` into this expression to get even closer.