Many of the functions in this homework assignment produce plot outputs. You have been provided a function called `studentPlotCheck` to check your plots against the solution function. DO NOT simply compare plots visually! There may be small issues that you can't catch just by looking. You can type `help studentPlotCheck` in the Command Window for information on how to use it. Also, there may be previous versions of this plot checker floating around the interwebs/your friend who took the class last semester, etc. Do not use these versions! Make sure you use the plot checker released with this homework.

Happy Coding!
~Homework Team

Function Name: `cellGrowth`

Inputs:

1. *(double)* A vector of cell counts
2. *(double)* A vector of time points

Outputs:

(none)

Plot Outputs:

1. A plot of cell growth vs time

Function Description:

Cell cultures undergo logistical growth until they have reached the carrying capacity of their environment. Given a vector of cell counts and the corresponding time plots, create a graph that helps visualize the growth of cells. You will also plot two lines, one representing the mean population size and the other representing the maximum population size. The requirements for producing the graph are as follows:

- Plot the cell count (y-axis) vs time data (x-axis) in red points
- Plot the mean population size in blue dashdots; use the same x-values as the main plot
- Plot the max population size in magenta dashes; use the same x-values as the main plot
- Both axes should range from ± 5% of the maximum and minimum values.
    - i.e. if the x-values provided range from 0-100, the x-axis range should be [-5 105]
- Use `axis square`
- Label the graph `'Cell Growth vs Time'`
- Label the x-axis `'Time'`
- Label the y-axis `'# Cells'`

Notes:

- Use `help plot` to see a list of formatting characters.
- The two input arrays will always be the same length.

Function Name: `airfoilz`

Inputs:

1. *(double)* A vector of angles of attack (will always be integers)
2. *(double)* A vector of lift coefficients
3. *(double)* A vector of angles at which to determine the lift coefficient value

Outputs:

1. *(double)* Lift coefficient at the specified angle

Plot Outputs:

1. Lift coefficient as a function of the angle of attack

Function Description:

As an engineer at Georgia Tech, you have been asked to analyze the lift coefficient for an airfoil. Anyone can be a rocket scientist!

There are a few different aerodynamic coefficients your typical aerospace engineer cares about. One is the coefficient of lift (the force pushing up on a wing) which changes as the angle of attack (the angle of the wing) changes).

You will be given a vector of angles of attack, and a vector of corresponding coefficients of lift. You must fit a 2nd degree polynomial to the data and plot the fit as a black line. When plotting the fit, you should calculate points at every integer x value between the minimum and maximum angle of attack, inclusive. On the same plot, you should plot the original data as blue stars.

You also want to determine the lift coefficients at particular angles of attack and output these values in a vector. The angles will be given in the third input and there may be any number of them. Use spline interpolation on the polynomial fit to compute the lift coefficients that correspond to the given angles.

Notes:

- All of the input vectors will be of the same length.
- Round your output to the nearest thousandth.
- For interpolation, use `interp1()` with the `'spline'` option.

Function Name: `proctorTest`

Inputs:

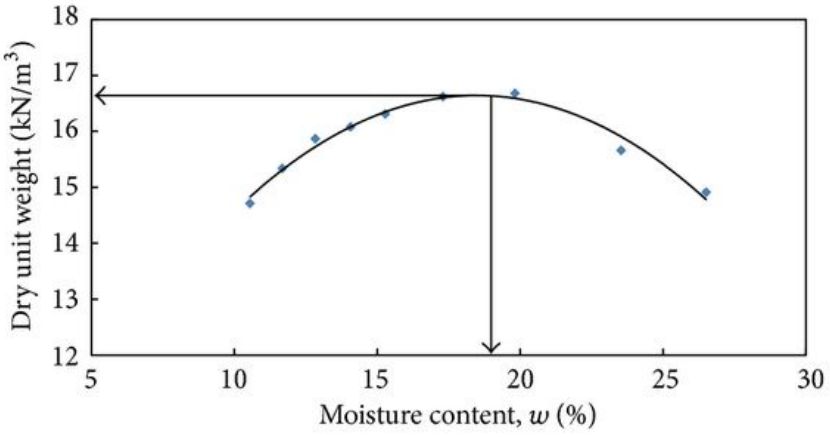
1. *(char)* The name of an Excel file filled with Proctor Test data points
2. *(double)* The "percentile of Standard Proctor", a percent of the maximum Dry Unit Weight

Outputs:

1. *(char)* A string containing the optimal water content and maximum dry unit weight
2. *(double)* The area beneath the curve above the percentile of Standard Proctor

Function Description:

Soil compaction is an important design consideration when designing large structures or building in unusual locations like beaches and marshes. The Standard and Modified Proctor tests are used to determine information about soil compaction. The tests are used to generate compaction curves which is a plot of dry unit weight vs. moisture content. (see below).



You need to find both the maximum dry unit weight (where the curve of the given points is maximal) and the corresponding optimal water content (water content value at the maximum dry unit weight). Once these numeric values are obtained, the first output should include them with their proper units in a string formatted as the one below:

`'<moisture content> <units>, <max dry unit weight> <units>'`

The units can be found in parenthesis of the Excel file table headers, always ordered as `'Moisture Content (<units>)'` in the first column, and `'Dry Unit Weight (<units>)'`

in the second column.

To find the maximum dry unit weight, you should numerically differentiate the given data, then find where the numerical derivative is zero by using spline interpolation. When taking a numerical derivative, you will end up with one less y value than you started with. You should assume that each of these values corresponds to the midpoint of the corresponding x values. For example:

```
x = [2, 3, 5, 6];
y = [1, -3, 5, 3];
```

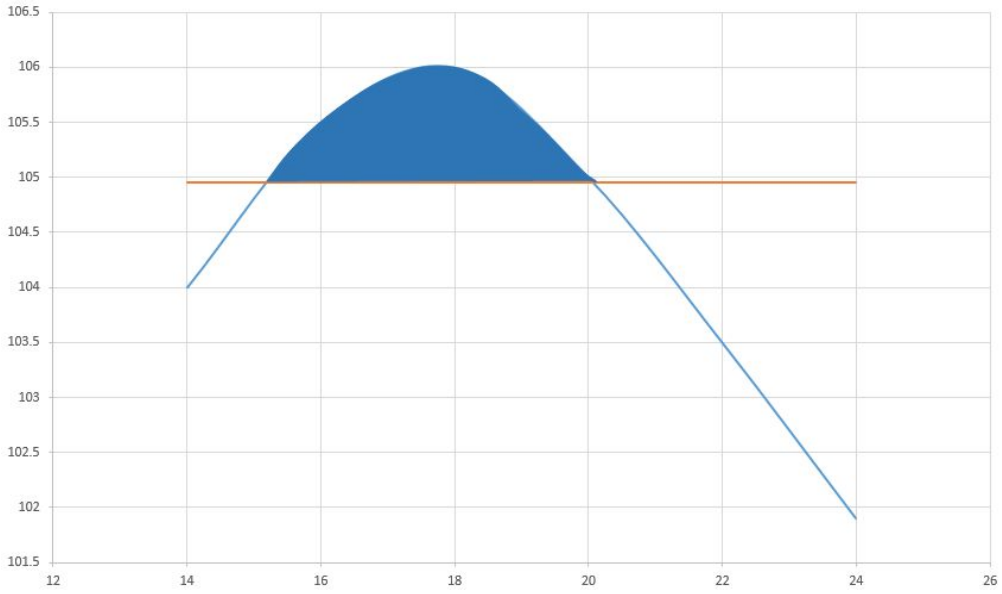
The y values of a numerical derivative for this data will be:

```
derivY = [-4, 8, -2];
```

And the corresponding x values should be:

```
derivX = [2.5, 4, 5.5];
```

Finally, the percentile of Standard Proctor will be given, which represents the acceptable level of compaction for a soil in the field. You must find the area above this percentile (usually 90 - 100) and beneath the curve. For example, if the maximum dry unit weight is 106 $\frac{lb}{ft^3}$ and a 99% standard proctor is given, the integral of the curve above the 104.94 $\frac{lb}{ft^3}$ horizontal line should be calculated (see below).



Notes:

- The `trapz()` function can be used to numerically integrate.
- All output numbers should be rounded to the thousandths place.
- 3 decimal places should be displayed in the output string.
- You can use `'%0.3f'` inside of `sprintf()` to display 3 decimal places.

Hints:

- To calculate the integral above a certain percentile, think about how you can shift the data such that a normal integral will be the area that you want.

Function Name: `wordDist`

Inputs:

1. (*char*) A string containing the name of a text file (with .txt extension)

Outputs:

1. (*char*) A string specifying if the text is at our reading level or not

Plot Output:

1. A histogram displaying the frequency of words of different lengths

Function Description:

"Students at Georgia Tech are just not as good as reading as students elsewhere." How many times have you heard that from the haters before? Well, if we get to choose what we read, then we can always just read something that is within our reading level. To facilitate that, we have to, of course, go to our trusty friend MATLAB!

Write a function called `wordDist` that takes in the name of a text file, and output a string and bar graph based on the count of words that are different length. If all the words are 13 letters or less, OR if the word "technology" (case insensitive) exists anywhere in the reading, the output should be

```
'We're at Georgia Tech, we can read that!'
```

Otherwise, the output shout be

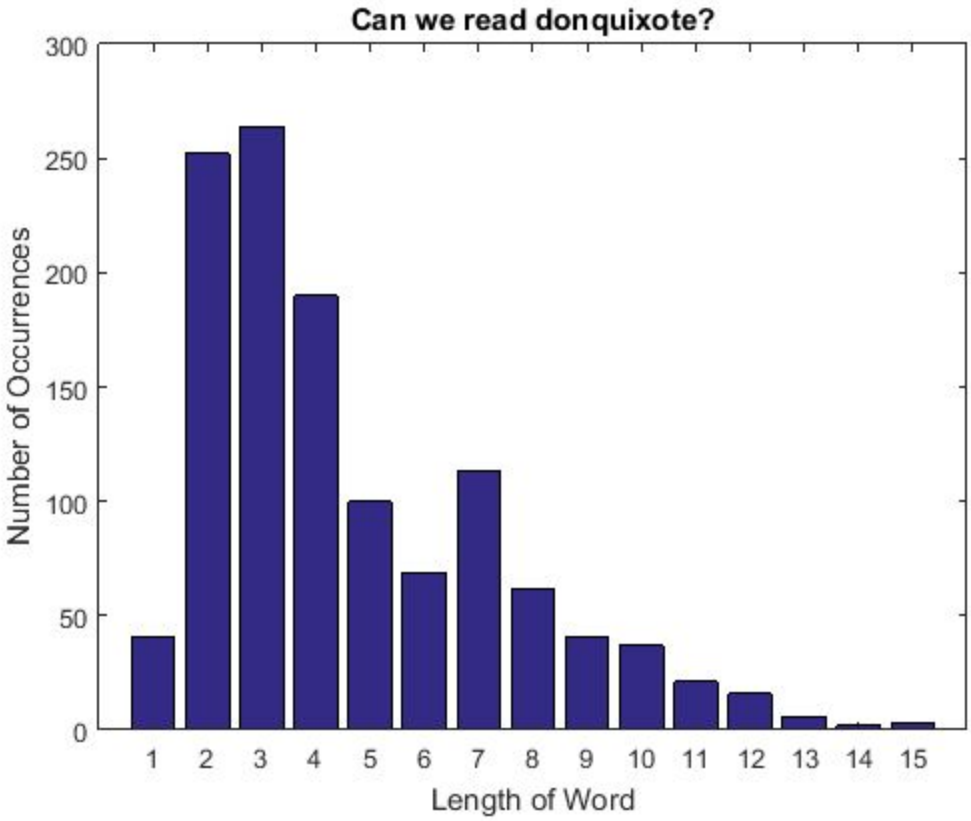
```
'We're at Georgia Tech, we can't read that :('
```

In addition, generate a bar graph of the length of words and their number of occurrences. Label the x-axis `'Length of Word'`; label the y-axis 'Number of Occurrences'; title the graph `'Can we read <file>?'`, where `<file>` should be the name of the input text file without the `'.txt'` extension. An example output plot is on the next page.

Notes:

- Use the `bar()` function to generate the bar graphs.
- Extraneous characters (non-letters) do not contribute to the word count.

Example plot output with input `'donquixote.txt'`



Function Name: `molecule`

Inputs:

1. (*double*) A vector containing the lengths of each consecutive line segment of the continuous molecule chain
2. (*double)* A vector containing the counterclockwise angles between each consecutive line segment and its previous line segment of the continuous molecule chain
3. (*logical*) A vector specifying which hexane rings to place circles within; will be an empty vector if molecule contains no hexane rings

Plot Output:

1. The plotted molecule

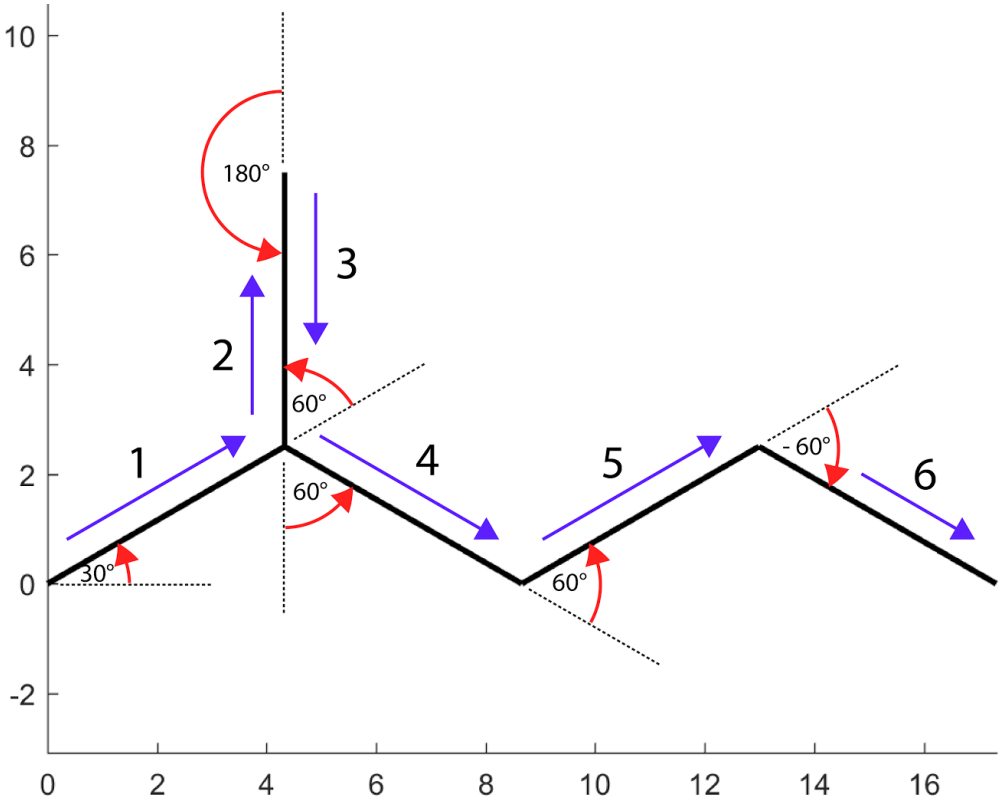
Function Description:

You've begun talking organic chemistry classes and are intrigued by molecule shapes. To make your organic chemistry homework easier, you decide to write a MATLAB function to plot molecule shapes such as this, described in detail below:

In order to simplify drawing complex molecule shapes, all molecule shapes will be treated as one continuous line. Imagine drawing a picture without raising your pencil. All molecules will consist of consecutive line segments. Your first input will be a vector specifying the lengths of each consecutive line segment in the molecule. Your second input will be a vector specifying the **counterclockwise** angles between consecutive line segments in the molecule.

In the following molecule, there are six consecutive line segments, highlighted as blue arrows:

```
molecule([5, 5, 5, 5, 5, 5], [30, 60, 180, 60, 60, -60], []);
```

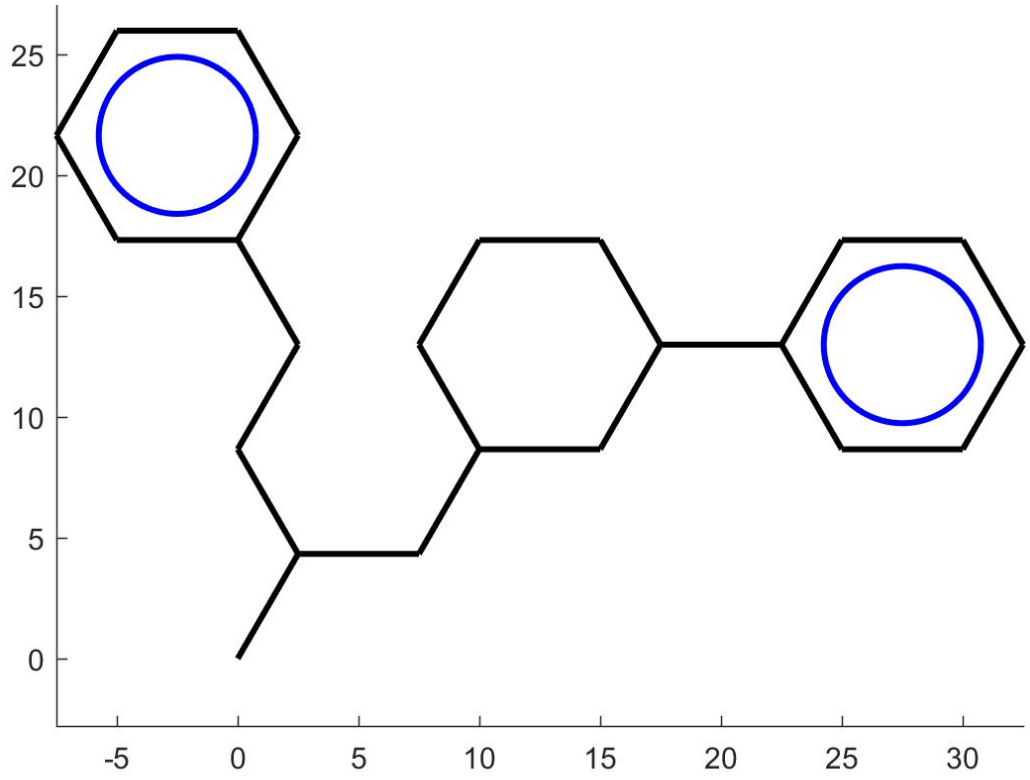


Each line segment has a length of 5. The blue arrows show the order in which the points are plotted. You see that some line segments overlap each other. The angles specified by the second input are also shown as the counterclockwise angles between consecutive line segments.

Certain molecules contain benzene rings, with circles drawn inside them to represent double bonds. You must plot circles within the benzene rings of the molecule specified by the third input. To help you with this, a helper function called `findCenter` has been provided. This function takes in an array of all the points of a plotted molecule in the format `[x1, x2, x3, x... ; y1, y2, y3, y... ]` and outputs the coordinates of the centers of each benzene ring in the molecule, as well as the radius of each benzene ring in the molecule. The third input of your `molecule` function will specify which hexane rings to fill with circles.

For example:

```
molecule(longMoleculeLengths, longMoleculeAngles, [true false true]);
```



For this molecule, after all the the points have been computed, the `findCenter` helper function will return benzene center points as `[-2.5000, 12.5000, 27.5000; <= x`
`21.6506, 12.9904, 12.9904] <= y`

Three benzene center points were found. The third input of `molecule` specifies that only the first and third benzene rings should be filled with circles. The radius of a circle plotted within a benzene ring must be 65% of the radius of that benzene ring. The circles should be composed of 100 points and be colored blue.

To rotate a set of points counterclockwise around the origin, multiply the coordinates by the rotation matrix as follows:

$$\begin{bmatrix} x_{rotated} \\ y_{rotated} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} x_{original} \\ y_{original} \end{bmatrix}$$

This is matrix multiplication, not element-wise multiplication!

You do not have to use rotation matrices to solve this problem, but it is given here if you would like to use it.

Notes:

- All molecules should begin plotting from the origin (`[0; 0]`).
- Line segments should be drawn black, and circles should be drawn blue.
- The first angle is measured from the x-axis.
- Both the rotation matrix and the `findCenter` helper function require input points to be in the format `[x1, x2, x3, x... ; y1, y2, y3, y... ]`, so it is advantageous to keep all points in your code in this format.
- You can type `help findCenter` in the Command Window for more information on the helper function.
- Use `axis square` and `axis off` for you plot output.
- You could theoretically draw anything you want with this function; there will be brownie points awarded for the coolest test cases.

Hints:

- Since the rotation matrix rotates points counterclockwise relative to the positive x axis, and the angle of each line segment is given counterclockwise relative to its previous line segment, try keeping track of each line segment's angle counterclockwise relative to the positive x axis. Then, the rotation matrix can be used to rotate and create any line segment.
- Making a few helper functions may be helpful to organize your thoughts/code.