Notice

This homework is designed to cover various topics from throughout the semester. It is for extra credit as well as to assist you in preparing for the final exam. However, it does not exhaustively cover all topics that you need to know for the final. You have until Wednesday, April 26th to submit it. To confirm, this homework is not mandatory and is only for extra credit.

Thanks for a great semester!
Good luck on Test 3 and the Final,
and as always,

Happy coding,
~Homework Team

Function Name: `chooseAdventure`

Inputs:

1. *(char)* The name of a text file with the start of your story

Outputs:

(none)

File Outputs:

1. A text file of the completed story

Function Description:

Have you ever spent time reading through the countless possible endings in a Choose Your Own Adventure book? You'll never have to do that with this problem - there's only one possible outcome for each story! The story starts at the beginning of the given text file, and continues until it reaches a tag in the format of:

`'<filename|line number|character number>'`

Such a tag indicates a jump to a different part of a different text file. For example, if the tag is `'<storyPartTwo.txt|5|8>'`, the story includes up to the character before the `'<'` and continues in the file `'storyPartTwo.txt'`on character 8 of line 5. Tags **may be** split across multiple lines of a the text file, such as:

```
Once upon a time, <storyPartThree.txt|
20|13> I learned how to use MATLAB
```

The completed story does not include any of the characters from the tag, including the angle brackets (`'<'`, `'>'`). When you start reading from a new file, you should start writing on a new line. So in the above example, if `storyPartThree.txt` contained the text `'there was a young prince.'` then the final file would read:

```
Once upon a time,
There was a young prince.
```

The story ends when it reaches the end of a text file.

The completed story must be saved in a file with the same name as the original in the function input with `'_chAdv'` appended before the `'.txt'`. For example, if the original input is called `'story1.txt'`, the output file would be named `'story1_chAdv.txt'`.

Notes:

- Line 1 means start on the first line of the text file, character 1 means start on the first character of the line.
- `'<'` and `'>'` will only be used for jumps to new story text files.
- You will never jump to a location between or on top of angle brackets.

- A story can jump to a new location in the same text file, and a story can jump any number of times.
- A story will not jump to a line that does not exist in a new file or to a character number that exceeds the number of characters in the line.
- Be careful with when you open/close files.
- You SHOULD include a single newline character at the end of your file. This is to match the solution file which prints a newline, not because it is specified in the problem description.

Function Name: `carShopping`

Inputs:

1. *(char)* An Excel file name
2. *(char)* Your first priority when buying a car
3. *(char)* Your second priority when buying a car

Outputs:

1. *(cell)* A sorted 4xM cell array of the best three cars

Banned Functions:

`sortrows()`

Function Description:

With your great resume and MATLAB skills, you've gotten an offer for an internship! Of course, you'll now need a car to get to work, and thankfully your parents are willing to give you a small loan. To decide on a car, you decide to write a function that sorts a spreadsheet of car data and outputs a cell array of the best three cars. The second and third inputs are the factors you consider the most important when selecting a vehicle, for example: year, mileage, or safety rating. You consider the second input to be much more important than the third so you should sort by that first. So if year is the most important factor and horsepower is the second most important factor, you will first sort by year. Then, within each year you will sort cars based on horsepower. After doing this, you will output the top three options to the final cell array. The first row of this cell array will have the original Excel file column headers and the remaining three rows will have the best three cars based on the input criteria, sorted from best to worst.

Notes:

- The top row of the spreadsheet will always be headers but are **not** guaranteed to be in a specific order.
- The categories you will be sorting by will always be quantitative, and in each category a larger value indicates a better car.
- In the case of a tie in both criteria, sort by order of appearance (`sort()` will do this automatically).
- All data is from carfolio.com.

Function Name: `dragRace`

Inputs:

1. *(double)* An NxM array of sampled points in time
2. *(double)* An NxM array of sampled velocities
3. *(cell)* A 1xN array of car names
4. *(double)* The specified race distance

Outputs:

1. *(char)* A string describing the race results

Function Description:

You always dreamed of being a racecar driver, but with your numerous other MATLAB projects, you haven't had the time to practice. Fortunately, the most prestigious auto race in the world is taking place this week: The MATLAB Grand Prix!

Using the given set of times and velocities, determine which car wins the race of given length and which car has the fastest acceleration. Each row corresponds to the data of one racer whose name is in the corresponding index in the cell array input.

Numerically integrate the given values to find the distance each car covers. Since each racecar will likely cross the finish line between two sampled points, linearly interpolate to find the times each car crosses the finish line. Then, numerically differentiate the velocities and determine which car has the fastest acceleration between two adjacent data points.

Output a string with the format:

```
'The <race winner> won the <race distance> meter race in <time> seconds! The <fastest acceleration car> had the fastest acceleration at <acceleration> m/s^2!'
```

Numbers in the output should be rounded to the nearest tenth. Be sure to check your answers against the solution file.

Notes:

- The times are given in seconds and the velocities are given in meters per second.
- The times and velocities array will have the same number of columns.
- The number of cars and the number of rows in the times and velocities arrays are the same.
- Remember to use linear interpolation between data points.
- You can use `'%0.1f'` as a format specifier inside of `sprintf()` to display a number to one decimal place.

Hints:

- `cumtrapz()` will be useful

Function Name: `shiaSurprise`

Inputs:

1. *(char)* The name of an image file with Shia LaBeouf
2. *(char)* The name of an image file picturing places around Georgia Tech's campus
3. *(double)* An integer indicating the range of values to replace

Outputs:

(none)

Image Outputs:

1. Edited .png image with `'shia_visits_'` prepended

Function Description:

Finals week(s) are nearly upon us and you're in need of a quick pep talk. After watching Shia LaBeouf's motivational video, you are super pumped and want other students to feel just as inspired. You decide the best way to do that is to virtually bring Shia to campus. To do this, you will replace the background color of the image of Shia with the corresponding pixels in an image of campus. If the images are different sizes, you should resize the campus picture to be the same size as the picture of Shia before performing the change.

To determine the background color, you should average the red, green and blue values found in the top left 10x10 pixel corner of the image. Once your have found the average RGB value (this is not a grayscale value; you need the average red, average green and average blue value), you will remove this value +/- the range given in the third input. Make sure all averages are `uint8` before applying the range. So you should replace any pixels where the red pixels fall within `redAverage+/-range`, green pixels fall within `greenAverage+/-range`, and blue pixels fall within `blueAverage+/-range`. All ranges are inclusive. If a pixel in the image of Shia matches this criteria, then that pixel should be replaced with the corresponding pixel in the image of campus.

The new image should be saved to `'shia_visits_<location>.png'` where `<location>` is the second input to the function, excluding the file extension.

Notes:

- Use `checkImage()` from HW12 to check your output.

Function Name: `bridges`

Inputs:

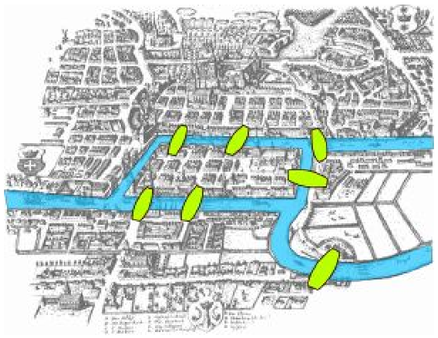
1. *(struct)* 1xN array of bridges

Outputs:

1. *(struct)* 1xM array of islands
2. *(cell)* Path or description of islands

Background:

You are Agent 007, AKA James Bond, and you have been assigned a new mission by Q. Your mission is to spy on a prominent Soviet mathematician to learn about the hugely important discovery it is rumored she has made. After a little romance, you learn that she has been walking around the islands and bridges of the city and come to the conclusion that it is impossible to go over all the bridges without crossing over a bridge twice! (see below).



Since this information is valuable to MI6, you decide to use your super-spy training and code a function in MATLAB to model this problem!

Function Description:

Let there be N bridges and M islands. The function should take in a 1xN structure array, where each structure represents an bridge. The fields of each structure will be the `Name`, `Endpoint1`, and `Endpoint2`. Each endpoint will be an integer representing an island. First, create a 1xM structure array, where each structure is one of the M islands. The islands should have a `Number` field containing their number and a `Bridges` field that contains a cell array of all the names of the bridges that have an endpoint on the island, sorted alphabetically. The island array should be sorted by number (so Island 1 is in the first index, then Island 2, etc.)

Then, determine if it is possible to cross every bridge without crossing one twice. If it is possible, output a cell array containing such a path. The cell at the nth index should contain the name of the bridge taken in the nth step of the path. Start with the bridge with the smallest value in the `Number` field and always use the bridge with the smallest `Number` value as the next bridge in the path. If it is impossible, output a 1xM cell array, where each cell represents the number of bridges that touch the island in the corresponding index of the first output.

Notes:

- If multiple bridges connect two islands, first use the bridge with the name that comes first sorted alphabetically (i.e the first one when the list of bridge names are sorted)
- It will always be possible to travel over bridges from any given island to another (the islands are connected).
- Islands are guaranteed to have distinct numbers and bridges are guaranteed to have distinct names.
- The provided algorithm is guaranteed to work on all sets of bridges and islands that will be provided.

Hints:

- Try inputting the original name of the city into the function for a fun easter egg!

Function Name: `subsetSum`

Inputs:

1. *(double)* A 1xN vector
2. *(double)* A single "target" number

Outputs:

1. *(logical)* Whether there is a subset of the vector that sums to the target number
2. *(double)* The subset, if any, that sums to the target value

Function Description:

Given a vector and a "target", determine if there is a subset of the vector that sums to the target number. The first output should be a logical of whether or not such a subset exists and the second output should be the subset. If no such subset exists, the first output should be false and the second output should be the empty vector (`[]`).

A target of 0 will always return true (selecting no elements sums to zero). Additionally, an empty vector as the first input will always return false (there are no elements to construct a sum out of).

Notes:

- You are guaranteed there will only be one unique subset that sums to the target.
- The only case where the first output is true and the second output is empty is if the target value is 0.

Hints:

- The last paragraph should give you a good idea of what base cases to use.
- To approach the base case, think about the following: you know that a single element in the input vector is either in the final subset or not. How can you use recursion to test if a particular element is in the final set?