Notice

This homework will require you to create functions that output spreadsheet files. These cannot be directly compared using `isequal`. However, we can compare them with a few lines of code!

```
>> functionName(in…);
>> functionName_soln(in…);
>> check = isequal(readcell('file.xlsx'),readcell('file_soln.xlsx'));
```

This homework will also have some plot outputs. We have provided you with the function `checkPlots()` to help you test the plot outputs of your functions. Use

```
help checkPlots
```

for a full description of how the function works, including what inputs to call with it. Below is the example from the documentation explaining how to run the function.

If you have a function called `testFunc` and the following test case:

```
testFunc(30, true, {'cats', 'dogs'})
```

To check the plot produced by `testFunc` against the solution function `testFunc_soln`, for this test case you would run:

```
[match, desc] = checkPlots('testFunc', 30, true, {'cats',
'dogs'})
```

Note:
Due to this homework spanning two different topics, it will not have the same difficulty progression that a typical homework does. Instead, The concepts have been split up by topic, High Level I/O being the first 3 and Plotting being the final 2. The problems are arranged in relative difficulty within the respective topics, however.

Happy Coding!
~Homework Team

Function Name: `genP`

Inputs:

1. (*double*) A number 1 - 9 representing the generation of Pokémon you are organizing

File Outputs:

1. A .xlsx file containing all the Pokémon from that generation

Topics: (*dir*), (*reading .txt files*), (*writing to .xlsx files*)

Background:

You're 32 and your parents are finally kicking you out of the house. As you are packing your boxes, you come across your old Pokémon card collection. The good memories of trading cards and battling the Pokémon come flooding back as you look through the cards. Once you are ready to pack them up, you realize just how disorganized the cards are. You don't know how you are going to possibly organize all of these cards. Then, you remember that you can use MATLAB to help you organize your cards by generation (the only respectable way to organize Pokémon cards).

Function Description:

Given the input, which represents the generation of Pokémon you are looking for, output a .xlsx file containing the names of all the Pokémon in the generation, as well as their HP (Hit Point) scores. Follow the steps below to generate the correct output:

- Find all Pokémon in your current directory that are a part of the inputted generation. All files representing Pokémon will be given as .txt files, and the filenames will be formatted as `'PokemonCard_<Pokémon name>.txt'` with a single underscore before the Pokémon name and generation number.
- In each .txt file representing a Pokémon, there will be several lines of text representing the base stats of the card. The **first line** of each .txt file will always represent the generation of the Pokémon, in the format `'Gen: <number>'`, where number represents the generation. If this number matches the inputted number, this Pokémon should be included in your output.
- Next, find the line containing the Hit Point (HP) value, which will be given in the following format: `'HP: <number>'`, where 'number' is the HP value. This statistic can be anywhere in the .txt file and is **NOT** guaranteed to be right after the first line. However, it is guaranteed that the line will be formatted as above, and it will not contain any additional text.
- After finding all the Pokémon in your directory that are part of a certain generation, sort the list of Pokémon names and HP by their names in alphabetical order.
- Output a .xlsx file that contains the names of all the Pokémon in the generation, as well as their HP values. The output file should be named `'generation<number>.xlsx'`.

Example:

% in current directory

```
PokemonCard_Pikachu.txt

1 | Gen: 1
2 | Type: Electr
3 | Att: 55
4 | HP: 35

PokemonCard_Eevee.txt

1 | Gen: 1
2 | HP: 55
3 | Att: 55

PokemonCard_Bayleef.txt

1 | Gen: 2
2 | Type: Grass
3 | HP: 35

genP(1)

generation1.xlsx
```

Eevee	55
Pikachu	35

Notes:

- The input value is guaranteed to be an integer between 1 and 9.
- There is guaranteed to be at least one Pokémon in each generation.
- Each card is not guaranteed to have the same number of lines in the .txt files. However, the lines containing the Generation number and HP value are guaranteed to be included in the file and will be formatted as above in each .txt file.
- Don't forget to close all .txt files that are opened!
- Be sure to be running this function in the **same directory** as all the files provided to you in the 'HW11.zip' file.

Hints:

- The function dir() might be useful

Function Name: `rocketRumble`

Inputs:

1. (*char*) filename of the Excel file containing your Pokémon
2. (*char*) filename of the Excel file containing your opponent's Pokémon

Outputs:

1. (*cell*) A 1x3 cell array containing the names of the top three Pokémon

File Outputs:

1. An Excel file containing the updated and sorted Pokémon data

Topics: (*High Level I/O*), (*Cell Arrays*), (*Masking*)

Background:

You're cleaning out your childhood bedroom with your best friend, and you stumble across all your old Pokémon cards! You each grab a handful of cards and challenge each other to see who has the best Pokémon in their hand. You turn to MATLAB to help rank your Pokémon cards and determine which three Pokémon between you and your opponent are the best!

Function Description:

You are given the filenames of two Excel files, each with four columns of data. The first file contains your Pokémon and the second file contains your opponent's Pokémon. Both files will contain columns with the following data: `'Name'` (*char*), `'Attack'` (*double, positive integers*), `'Defense'` (*double, positive integers*), and `'Type'` (*char*). These columns **are not guaranteed to appear in any particular order**, but the order of the columns in both files will match. The type of a Pokémon will always appear as `'Fire'`, `'Water'`, or `'Grass'`. Begin by combining your and your opponents data.

Create a new column with the header `'Total Score'`. Every Pokémon's score should initially be zero. You will need to modify each Pokémon's total score by following the guidelines below in this order:

1. If a Pokémon's attack score is...
    a. an **even** number, multiply their attack score by 2 and add this to their total score.
    b. an **odd** number, multiply their attack score by 1.5 and add this to their total score.
2. If a Pokémon's defense score is...
    a. **at least 50**, then add their defense score to their total score.
    b. **less than 50**, then subtract 10 from their total score.
3. If the Pokémon is...
    a. **Fire** type, add 25 to their total score.
    b. **Water** type, divide their total score by two.
    c. **Grass** type, take the square root of their total score.

Make sure to round the `'Total Score'` column to two decimal places. Append the `'Total Score'` column to the end of the initial data set. It should be the fifth column. Sort the Pokémon (array rows) by the `'Total Score'` column, from highest score to lowest score. Lastly, output the updated data with headers in an Excel file. Using the first input as the base filename, name the file in the following format `'<filename>_updated.xlsx'`. Also output a 1x3 cell array containing the names of the top three Pokémon. Output the names of the three pokemon with the highest scores in a 1x3 cell array

Example:

`myPoke.xlsx`:

Name	Type	Defense	Attack
Bulbasaur	Grass	49	50
Charmander	Fire	52	43
Squirtle	Water	48	65

`opponentPoke.xlsx`:

Name	Type	Defense	Attack
Ivysaur	Grass	63	62
Charizard	Fire	78	84
Blastoise	Water	100	83

```
bestPoke = rocketRumble('myPoke.xlsx', 'opponentPoke.xlsx')
```

`myPoke_updated.xlsx`:

Name	Type	Defense	Attack	Total Score
Charizard	Fire	78	84	271
Charmander	Fire	52	43	141.5
Blastoise	Water	100	83	112.25
Squirtle	Water	48	65	43.75
Ivysaur	Grass	63	62	13.67
Bulbasaur	Grass	49	50	9.49

```
bestPoke → {'Charizard', 'Charmander', 'Blastoise'}
```

Notes:

- The columns will not always appear in the same order in different test cases. However, the order of the columns for both input files in a given test case will match.
- The column headers will always appear as `'Name'`, `'Attack'`, `'Defense'`, and `'Type'` (order not guaranteed).
- You should use the first input (your Pokémon filename) to determine the output filename.
- There will always be at least three Pokémon in total, and each file will contain at least one Pokémon.

Hints:

- The functions `cell2mat()` and `num2cell()` will be helpful.
- Consider the use of vector operations and masking
    - this problem can be done without iteration (but does not have to be)

Function Name: `gottaCatchEmAll`

Inputs:
1. (*char*) filename of a .txt file

Outputs:
1. (*char*) a sentence describing what you caught

File Outputs:
1. An .xlsx file that contains the amount of times each word was said

Banned Functions:
```
split(), strsplit()
```

Topics: (*excel writing*), (*line-by-line, word-by word reading*), (*string tokenization*)

Background:

As you sit down to begin watching the 23rd season of Ash Ketchum's journey to be the very best, like no one ever was, you begin to wonder. Will he ever reach his 11th birthday? Do people eat pokemon? And can you use MATLAB to analyze scenes from the show? Unfortunately the answers to the first 2 questions are beyond the scope of human knowledge, but you do know the answer to the third question. The answer is "yes", yes you can.

Function Description:

Given the filename of a .txt file, count the number of occurrences of each unique word in the file. All punctuation should be ignored and not included in the solution file. Unique words are case insensitive and separated by spaces, but the singular and plural versions of words should be counted separately. For example, `'POKEMON'` and `'pokemon'` would be counted as the same word; however, `'pokemon'` and `'pokemons'` are counted as two different words. You should also not consider words contained within other words. For example, `'poke'` should not be counted when you come across the word `'pokemon'`. The data should be written to a .xlsx file named with the format

```
<original filename>_counted.xlsx
```

in which the first column contains the unique word in it's lowercase form and the second column contains the number of times the word appears in the text file. The words should appear in the .xlsx file in the order that they **first** appear in the text file.

Finally, you should output a sentence describing what you found:

```
'I caught 'em all! There were <num_unique_words> unique words,
 with the word '<most_common_word(lowercase)>' appearing
          <greatest_num_occurrences> times.'
```

Example:

scene1.txt

```
1| Pika.
2| Pi.
3| Pikachu.
4| Well, Pikachu, we're here.
5| It's a Pokemon paradise.
6| -This looks like tons of fun. -Yeah, the Pokemon love it.
7| Just keep your eye on Togepi, okay, Pikachu?
```

```
txtFn= 'scene1.txt'
countedSent = gottaCatchEmAll(txtFn)
```

countedSent→ 'I caught 'em all! There were 27 unique words, with the word 'pikachu' appearing 3 times.'

scene1_counted.xlsx

pika	1
pi	1
pikachu	3
well	1
we're	1
here	1
it's	1
a	1
pokemon	2
paradise	1
this	1
looks	1
like	1
tons	1
of	1

fun	1
yeah	1
the	1
love	1
it	1
just	1
keep	1
your	1
eye	1
on	1
togepi	1
okay	1

Notes:

- All words will be separated by a single space after punctuation is removed.
- All punctuation and special characters (except for apostrophes, spaces, and letters) should be removed when reading lines and evaluating unique words.
- The excel file should contain the lowercase version of the word regardless of the capitalization contained within the .txt file
- There will never be a tie in the number of occurrences of the most common word.
- Remember to use two single quotes when displaying an apostrophe in a string. (`'gotta catch ''em all'`).
- Words contained within other words should not be counted
- **The most common word should be in single quotes in the output string!**
- The most common word should be lowercase in the output string

Function Name: `pokemonGO`

Inputs:

1. (*double*) 1XN Vector of timestamps at which you caught pokemon
2. (*double*) 1XN Vector of positions at which you caught pokemon
3. (*double*) Timestamp at which your sibling took your phone

Plot Outputs:

1. The graph of all the positions vs. timestamps of when you caught pokemon

Topics: (*plotting)*

Background:

The year is 2016 and you just got home from school. You're eager to walk all around the neighborhood catching only the finest, most unique pokemons. But when you open your phone, you realize someone used all your pokeballs to catch a surprising number of Fearows, one of the weakest pokemon in the entire game! You're livid when you realize that your sibling used your phone to waste all your pokeballs. But you won't stand for this, so you use MATLAB to see how your stats looked before your sibling stole your phone!

Function Description:

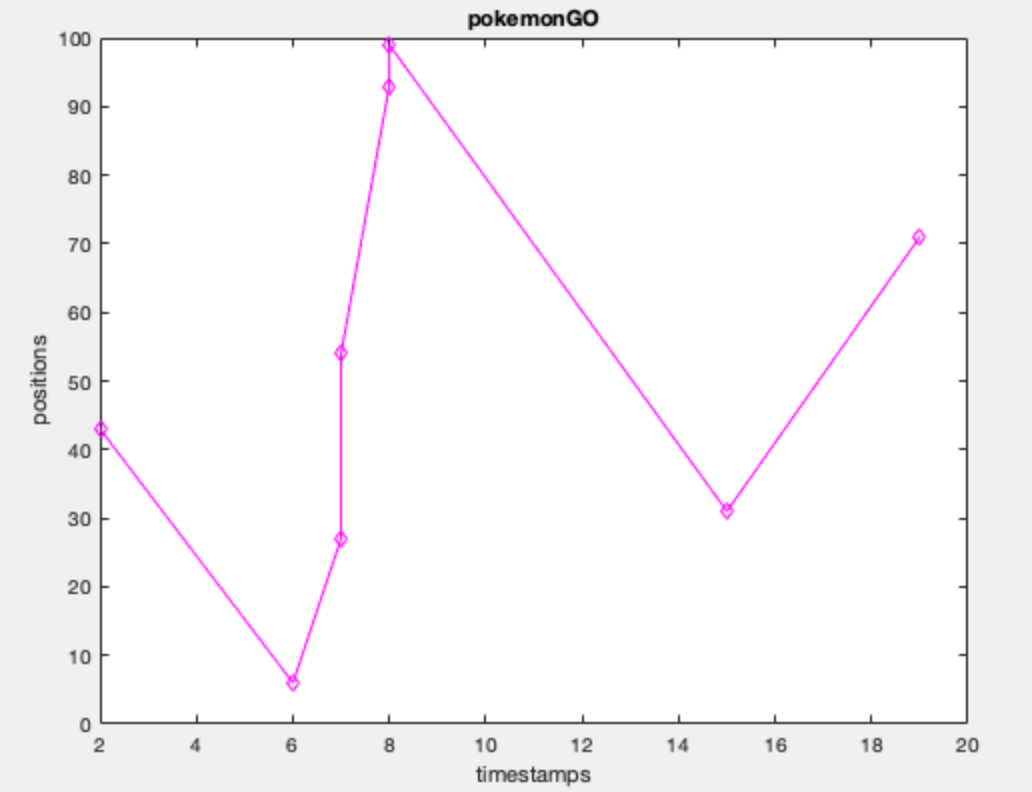
First, remove all the time values and corresponding position values that are greater than the timestamp at which your sibling stole your phone. Next, on a **square** graph, plot the remaining positions vs. timestamps with a magenta solid line that has diamond points. Title your graph `'pokemonGO'` and your x and y axes `'timestamps'` and `'positions'` respectively.

Example:

```
time = [2, 6, 7, 7, 8, 8, 15, 19, 23, 27, 27, 35, 36, 40, 40]
pos = [43, 6, 27, 54, 93, 99, 31, 71, 55, 96, 42, 70, 95, 74,
67]
cutOff = 20

pokemonGO(time, pos, cutOff)
```

Plot: (on next page)



Notes:

- The original positions and timestamps vectors are guaranteed to be the same length
- The positions and timestamps vectors are guaranteed to have positive values
- There will always be at least 1 point to plot

Function Name: `pokemonRanger`

Inputs:

1. (*char*) File name of an excel sheet with information about Pokémon's locations

Plot Outputs:

1. Figure with 1x3 subplots depicting the captured Pokémon

Banned Functions:

`polyshape, nsidedpoly`

Topics: (*reading .xlsx files*), (*plotting shapes*), (*conditionals*)

Background:

As a child, you were never really into that whole Pokémon trainer deal. Throwing a ball at a Pokémon and then having it be your loyal companion forever and ever? There's no fun in that! No, you've always aspired to have the more rewarding job, the job where instead you draw weird shapes around Pokémon and have them join you temporarily to complete arbitrary tasks. You choose to become a Pokémon Ranger! Unfortunately, you're terrible at drawing shapes and so you're failing Ranger School. To help you succeed, you turn to your truly loyal companion: MATLAB!

Function Description:

You are given a string containing the name of an excel file which holds information about several wild Pokémon. The file will always contain five columns: `'Name'` *(char)*, `'Type'` *(char)*, `'X Location'` *(double)*, `'Y Location'` *(double)*, and `'Size'` *(double)*, **given in this order**. The first row of the file will contain column headers, and each subsequent row will correspond to one Pokémon. The `'Type'` column will contain a string with the Pokémon's type, which is guaranteed to be either `'Grass'`, `'Fire'`, or `'Water'`. The `'Size'` column is guaranteed to always contain a positive double.

It is your job to generate a figure of 1x3 subplots on which you will plot the location of each Pokémon and draw a corresponding shape around it based on the Pokémon's type. Use the following convention to determine what subplot to plot on, and what shape to draw:

1. If the Pokémon's type is `'Grass'`, plot its location using a **green star** on the **first** (left-most) subplot. Then, plot a **green circle** centered on the Pokémon's location, whose **radius** is the Pokémon's size.
2. If the Pokémon's type is `'Fire'`, plot its location using a **red pentagram** on the **second** (middle) subplot. Then, plot a **red square** centered on the Pokémon's location, whose **side length** is the Pokémon's size.
3. If the Pokémon's type is `'Water'`, plot its location using a **blue dot** on the **third** (right-most) subplot. Then, plot an equilateral **blue triangle** whose centroid (triangle center) is at the Pokémon's location, and whose **side length** is the Pokémon's size.

Finally, set the axes of all your subplots to `axis equal`.

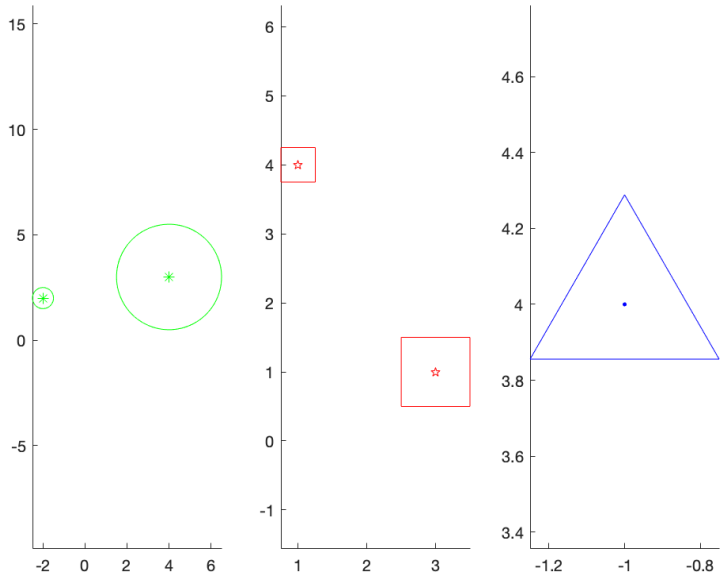
Example:

`wildPokemon.xlsx:`

Name	Type	X Location	Y Location	Size
Treecko	Grass	-2	2	0.5
Piplup	Water	-1	4	0.5
Cyndaquil	Fire	1	4	0.5
Charmander	Fire	3	1	1
Venusaur	Grass	4	3	2.5

```
pokemonRanger('wildPokemon.xlsx')
```

`plot →`



Notes:

- You should plot each Pokémon in the order in which it appears in the Excel file (i.e. Treecko was plotted first and Venusaur was plotted last in the example above)
- To plot the circle, use 100 linearly spaced values for theta between 0 and 2π

- You are guaranteed to find at least one grass, fire and water type Pokémon
- To make `checkPlots` happy, you must set your axis after every call to `plot`
-

Hints:

- Think of how you could use equilateral triangle trigonometry to determine the x and y-coordinates of the vertices knowing only the side length and location of the centroid (triangle center)