Notice - Testing plot outputs:

For this homework, some of your outputs will be plots. 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:

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

Happy Coding!
~Homework Team

Function Name: `pokemonAthon`

Inputs:

1. (*double*) A 1xN vector describing the instantaneous velocities of your Ponyta
2. (*double*) A 1xN vector describing the times at which those velocities occur

Outputs:

(None)

Plot Outputs:

1. A single figure with 4 subplots showing the position, velocity, acceleration, and jerk

Banned Functions:

`polyfit(), polyval(), polyint(), polyder()`

Background:

You have just gotten you first Ponyta and you want to enter into the 2019 Flame Pokémon-athon, the most competitive Pokémon race out there for trainers to test the speed of their fire-type Pokémon! In order to be the very best you can be, you measure the instantaneous velocity of your Ponyta at different points in time along then analyze your Ponyta's performance.

Function Description:

Given a vector of velocities and a vector of times, calculate your Ponyta's position (integral of velocity with respect to time), acceleration (derivative of velocity with respect to time), and jerk (2nd derivative of velocity with respect to time) for the duration of the race. Then, plot your results in a 2x2 subplot. You should make all of your calculations using **numerical** integration and differentiation. Each subplot should have its axes labeled with the name of the quantities being plotted (e.g Position, Velocity, Acceleration, Jerk, Time), and contain the following results in the specified style

1) Position versus Time, green solid line
2) Velocity versus Time, red solid line
3) Acceleration versus Time, blue solid line
4) Jerk versus Time, black solid line

You also need to specify the length of each axis. The range of each X axis should be from the minimum time to the maximum time for that subplot and the range of each Y axis should range from the minimum value minus 1 to the maximum value plus 1.

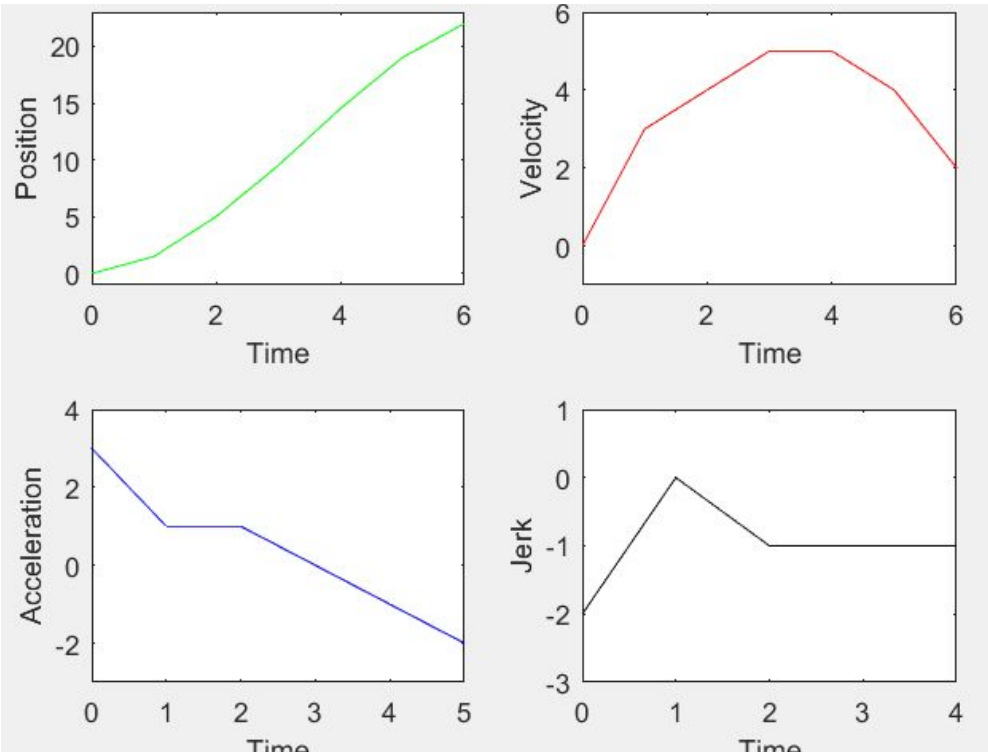
Notes:

- You should use `cumtrapz` for integration and `diff` for derivation.
- Your inputs are guaranteed to be at least length 4.
- Shorten the time vector for derivatives by removing the last element(s).
- The data given is guaranteed to be in time order.

Continued...

Example:

```
pokemonAthon([0 3 4 5 5 4 2],[0 1 2 3 4 5 6])
```



Function Name: `safariZone`

Inputs:

1. (*double*) A 1xN vector describing the coefficients of a polynomial
2. (*double*) A 1x2 vector describing the interval over which to plot the polynomial

Outputs:

(None)

Plot Outputs:

1. A plot describing the Dratini population in the Kanto region over time

Banned Functions:

`polyint(), polyder()`

Background:

You trek to the Safari Zone in Kanto, located just north of Fuchsia City in search of the vast colony of Dratini. In your backpack, you have your Pokédex, but alas! The Pokédex has powered down, and you are left without its crucial data. How will you find the Dratini now?

Before the Pokédex shut off, you were able to scribble down in your notebook the polynomial describing the the population of Dratini in the Kanto region over time. Use MATLAB to restore the rest of the information to help you in your search of Dratini!

Function Description:

Write a MATLAB function that takes in a vector of polynomial coefficients and an interval to plot over and creates a single plot showing the polynomial, its derivative, and its integral.

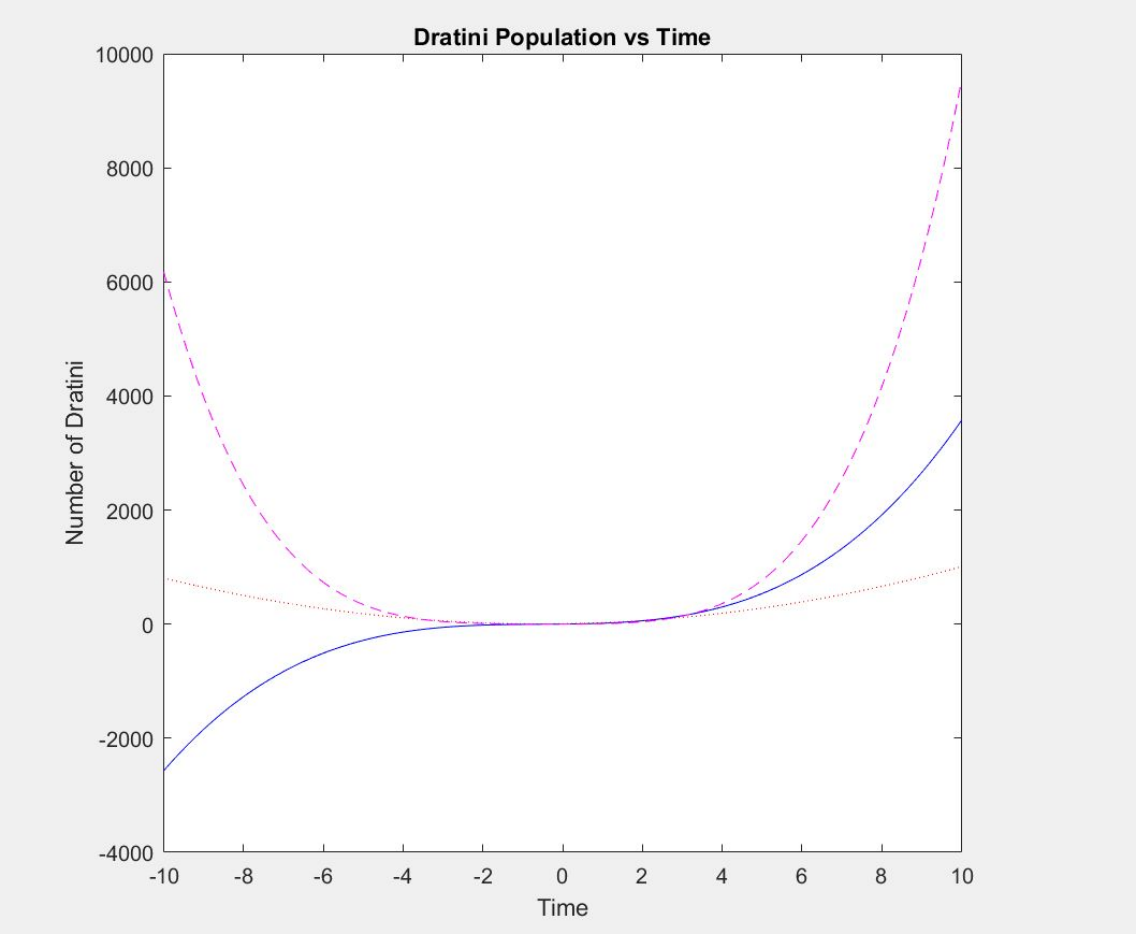
Notes:

- Use the y-intercept as the constant of integration.
- You should use 100 evenly spaced points along the x-axis to plot your three lines.
- You should use a blue solid line for the original polynomial, a red dotted line for the derivative, and a magenta dashed line for the integral.
- Title your graph `'Dratini Population vs Time'`.
- Label the x and y axes `'Time'` and `'Number of Dratini'`, respectively.
- After plotting, set your axes to `'square'`.

Continued...

Example:

```
safariZone([3 5 7 1], [-10,10])
```



Function Name: `levelUp`

Inputs:

1. *(double)* Nx7 Array of Pokémon statistics at previous levels
2. *(double)* Desired level of the Pokémon

Outputs:

1. *(double)* 1x7 vector of Pokémon statistics at desired level

Background:

You are Brock. Your friend Ash Ketchum has dreams of becoming the greatest Pokémon trainer. As an aspiring breeder, you need to help him out by making sure his Pokémon have top-tier stats. You've measured their stats throughout the journey, but you want to be able to predict what their stats will end up at, or what other Pokémon's stats might be. You decide to use MATLAB to help him out!

Function Description:

You will be given an array of a Pokémon statistics in which the first column represents a level and the second through 7th columns represents the numerical statistics corresponding to that level. The second input is the level at which you want to calculate the new statistics. Use linear interpolation, or extrapolation if necessary, to determine the new statistics, and output a vector of the statistics in the same order, with the desired level concatenated to the left.

Example:

The level values in the first column of the input and output are bolded. In this case, the value of the second column in the output is the linearly interpolated value at level 7 of the statistic values in the second column. The value of the third column is the linearly interpolated value at level 7 of the statistic values in the second column, and so on.

```
pikachu = [ 50   111    82    60   102    63   142;        level = 7
            25    60    44    32    53    33    72;
            5     20    13    10    11     9    15]

out = levelUp(pikachu, level)
out → [7    24    16    12    15    11    21]
```

Notes:

- Round the output to the nearest integer.

Function Name: `rootChopper`

Inputs:

1. *(double)* A set of x data
2. *(double)* A corresponding set of y data
3. *(double)* A lower bound on the x values
4. *(double)* An upper bound on the x values

Outputs:

1. *(double)* The x position of a root

Background:

Along your Pokémon adventures there were numerous obstacles, from the many different gyms to the fields of wild Pokémon; however, there was never so great an obstacle as the lone tree in your path. Unfortunately, the only Pokémon you have who knows cut is a little slow on the uptake and has missed the tree several times. You, being the MATLAB wizard/Pokémon trainer that you are, decide to make a function to find exactly where the roots of the tree are so that your Pokémon has no excuse to miss again!

Function Description:

The bisection method is a root finding algorithm that, given an interval [a,b] that contains only one root, iteratively cuts that interval in half, keeping the half that contains the root.

1. Given a set of x and y values, determine the coefficients of the highest order unique polynomial that passes through all the points.
2. Calculate the midpoint of the interval using the following formula:
$$x_{midpoint} = \frac{lowerBound + upperBound}{2}$$
3. Calculate the function values at the lower bound, the upper bound, and the midpoint using the coefficients determined in step 1.
4. If the sign of the function value at the midpoint is opposite of the sign of the function value at the lower bound, replace the upper bound with the midpoint.
5. Otherwise, if the sign of the function value at the midpoint is opposite of the sign of the function value at the upper bound, replace the lower bound with the midpoint.
6. Repeat steps 2 through 5 until the absolute value of $\frac{upperBound - lowerBound}{upperBound + lowerBound}$ (the approximate error) is less than 0.0001. Your final guess for the root will be the midpoint of the final interval.

….Continued

Notes:

- Round your **final** answer to two decimal places.

Hints:

- If the product of two numbers is negative, then they have the opposite sign.

Function Name: `whosDatPokemon`

Inputs:

1. (*double*) A 1xN vector containing the lengths of each consecutive line segment of the Pokémon
2. (*double)* A 1xN vector containing the counterclockwise angles in degrees between each consecutive line segment and its previous line segment

Plot Output:

1. The plotted Pokémon

Background:

You have taken an interest in guessing Pokémon by their outlines and wonder if MATLAB can do it for you. Just like the "Who's that Pokémon!" minigame, you will create an outline of a Pokémon as specified by the inputs.

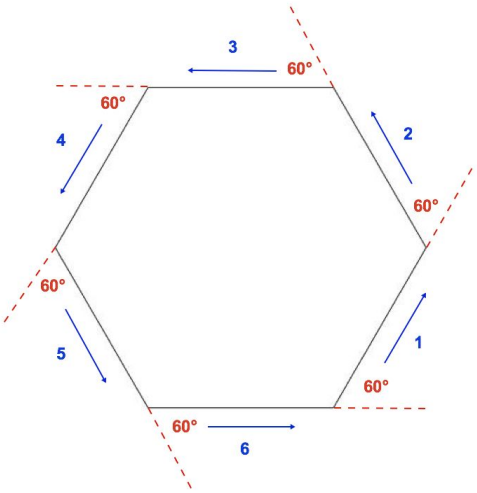
Function Description:

This function will take a vector of line segment lengths and angles and plot a Pokémon. The drawing will start at (0,0) and each line segment will be drawn from where the last one ends. Your first input will be a vector specifying the lengths of each consecutive line segment in the Pokémon. Your second input will be a vector specifying the *counterclockwise* angles between consecutive line segments. Draw the Pokémon with a black line.

Example:

The following input to `whosDatPokémon` will result in the hexagon below:

```
>> whosDatPokemon([5, 5, 5, 5, 5, 5], [60, 60, 60, 60, 60, 60]);
```



...Continued

Notes:

- The first angle is measured from the x-axis.
- After plotting, set your axes to `'square'`.
- You could theoretically draw anything you want with this function; See if you can come up with some cool test cases!
- The first test case is a simple hexagon, but the other 3 are Pokémon! Grading test cases may not necessarily be Pokémon.

Hints:

- Try doing this on a whiteboard first. How do the x/y coordinates of the next point change at each step?
- The `cumsum()` function may be useful. How would you use it on the hexagon example?
- Remember that to plot a line in MATLAB, you need a starting point AND an ending point. You may need to keep track of the point you plotted in the previous iteration.