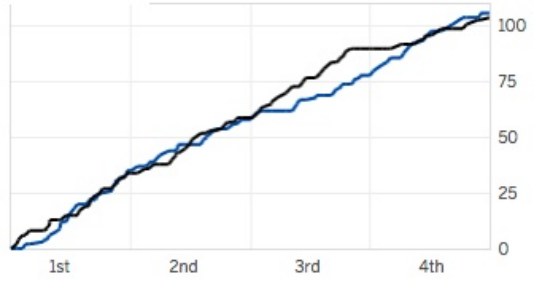
Exercise 8 - Using control structures in Python

Complete the tasks below and submit your results via a pull request on GitHub by the beginning of tutorial next Friday.

To begin this week, fork the TA's Exercise 8 Github repo. Clone the forked repo so that you have the required files. Be sure to commit regularly to demonstrate incremental progress on the assignment.

1. Analysis of data surrounding sports teams has grown into a major business for the teams themselves and the media. One cool summary plot that media outlets generate to summarize a game, in this case basketball, is a line graph depicting the cumulative score for each team as a function of time in the game (see below).



I first saw this plot a number of years ago when reading about the results of a game where the University of Wisconsin (Go Badgers!) played Michigan State University. Using the score-by-score information from this game summarized in "UWvMSU_1-22-13.txt" generate a graph similar to the one I show above. Don't worry about how pretty your graph is. Focus more on the control structures required in your script used to generate the plot.

A couple tips on this one:

- You'll want to generate a dataframe with a cumulative score for each team whenever either team scores.
- For plotting purposes, keep it simple. The easiest way to make a plot with two lines is to use the `plot()` function in the `matplotlib.pyplot` package. You'll need a line in your script to import the package (`import matplotlib.pyplot as plt`). Then when you want to plot use the `plot` function with the following syntax `plt.plot(time,UWscore,'r-',time,MSUscore,'g-')`, where time is the time column of your array or dataframe and UW and MSU refer to the cumulative scores for each time in your array or dataframe.`r-` and `g-` indicate solid red and green lines, respectively.

2. Write a game called "guess my number". The computer will generate a random number between 1 and 100. The user types in a number and the computer replies "lower" if the random number is lower than the guess, "higher" if the random number is higher, and "correct!" if the guess is correct. The player can continue guessing until they get it right.

Here's an example game where the random number is 79. User inputs are *italicized*.

```
I'm thinking of a number 1-100...

Guess: 40

Higher

Guess: 70

Higher
```

`Guess:` *80*

`Lower`

`Guess:` *77*

`Higher`

`Guess:` *79*

`Correct!`

A couple tips on this one:

- take a look at the Input/Output reference I gave you for how to get input from the user
- `numpy.random.choice()` is a function that allows for a random selection from a 1D array; you'll have to import `numpy` to use this function
- beware of "infinite loops" on this one. To end such a problem, in IPython and Spyder type ctrl-c to break you out of the loop.

Turning in your assignment via GitHub

Once you have committed all changes to your local Git repos and pushed all of those commits to the forked repo on GitHub, you can "turn in" your assignment using a `pull request`. This can be done from the GitHub repo website. When viewing the forked repo, select "Pull requests" in the upper middle of the screen, then click the green "New pull request" button in the upper right. You'll then see a screen with a history of commits, select the green "Create pull request button". In the text box next to your user icon near the top of the page, remove whatever text is there and add "owner's last name submission", but obviously substitute your last names. Then click the green "Create pull request" button.