Jason Woo
Hanna Yoo
Joeleen Moy
MHC Computing Concept
Final Project Proposal

For our final project, we're hoping to create our own version of a Markov text generator based off a scene from "Bruce Almighty." In the movie Jim Carey's character changes the text on the teleprompter, turning a normal broadcast into chaos. We wanted to recreate this using a text analyzer and generator to create our own news teleprompter that seems to work at first, but slowly reverts into nonsense.

Our goal for the project is to create a working code that will act as a "teleprompter," generating a fictional news story that looks real. Our stretch goal is to add the Markov Almighty factor. After the regular text generator function runs for a while, the teleprompter should have a "glitch" where the letters in a few words are rearranged randomly to create nonsense. Slowly, more words will be rearranged until every output is gibberish.

Steps:

1. Inputting and outputting files
2. Text analysis (from news cast dictionary)
3. Text generation
4. Add "randomizer" that will slowly start shuffling letter order in all the words until each output is complete gibberish.

See the clip from the movie: (https://www.youtube.com/watch?v=NSWGRfktwmQ)

Jason Woo
Joeleen Moy
Hanna Yoo
MHC Computing Concept
Final Project Reflection

Inspired by 'Bruce Almighty', we were able to create a Markov Almighty that mimicked Steve Carell. Through teamwork and some help from the CS masters, we were able to hit our goals as well as our initial stretch goals. However, ideas on how to improve and add to the application are limitless.

We were drawn to the idea of creating a Markov text generator because it seemed challenging to create a program that would be able to “live” and “think” on its own. Adding the extra dimension of a randomized teleprompter made the project challenging and our own. We were able to add some comedy and character to our code. As we progressed, we began to understand more and more the flexibility and possibilities that coding offers.

To start our project, we used the initial code from Harvey Mudd College’s homework #11. We also took some transcripts from 'Anderson Cooper 360 Degrees' mostly regarding stories about the GOP and presidential candidates. After copying and pasting the transcripts, we converted them into a text file. We defined text, wordcount and mk with ease. We ran into our first obstacle trying to define generateText. We had trouble with the random.choice figuring out how Markov would know which words to randomize. Our next obstacles were our stretch goals. We initially struggled to make Terminal act as a “teleprompter.” We had wanted to include a time.sleep when Markov generated new text. We also wanted to limit the number of words per line for a cleaner look. We had trouble creating this effect and implementing a time.sleep function within a loop and having it output properly. Initially, having the time.sleep within the loop wouldn’t create an output at all, or would wait for the entire output to print after a long period of time. Taking the Professor's thoughts into consideration, we included a random.choice in our

time.sleep that had a variety of different times that Markov would implement. We also used %(mod) to assign 10 words per line.

At the start of running the project, it will ask you the name of the text file you would like to use. Input the filename (must be in .txt format) and the project will run on its own. The computer will then output an actual news story, then generate a random news story, outputting it line by line as a teleprompter would. After a while, the words become scrambled and the computer would just output nonsense. Though creating a program that would purposely “glitch” and break seemed counterintuitive, we thought this added character to the code and made Markov our own. This three step function acts as a Markov text generator and a Bruce Almighty teleprompter.

We can improve our project by combining an existing grammar checker to the source code. This will make the text from the Markov randomization process more coherent. Another improvement that can be made is the length of the text outputted/generated. The text file used for the demo is 20,000+ words, so we outputted the first 76 words from the original file, 1% using Markov, and 1% using Markov and letter randomization. To solve the problem, we can ask the user to specify the proportion or word length he/she wants outputted. The program has many different applications, though not all completely useful with the “Bruce Almighty” factor. The program can be used to pull any plain text file, making the output variations endless. The Almighty factor can also be commented out so the code can function as a simple text generator.

Overall, we were able to create a program that we are very proud of. Our version of the Markov text generator injects the traditional text generator with some comedy and random nonesense os we kithn ti swa revy oogd. Stkans orf a ogod emtrster! . . .