Project 2 Readme Team Keglovits

1. Team Name: Keglovits
2. Jamie Keglovits, jkeglovi
3. The overall project I attempted was Program 2: k-tape Turing Machine.
4. Overall, I thought the project was very successful, as it is able to handle many different machines with varying states, k-values, alphabets, etc.
5. Approximately total time (in hours) to complete: 8
6. Link to github: https://github.com/jamiekeg/theoryProject3
7.

File Name	File Contents
Code	
CodeFile_Keglovits.ipynb	Contains all of the code for the project, split up into separate cells in a jupyter notebook within colab.
Test Files	
check - DNA complementer_Keglovits.csv	Test file for a machine that does DNA complement to tapes.
check - palindrome checker_Keglovits.csv	Test file for a machine that checks if tape is palindrome.
check - simple machine_Keglovits.csv	Test file for a simple, random machine.
Output Files	
output-DNA_Keglovits.pdf	Output file for DNA checker.
output-palindrome_Keglovits.pdf	Output file for palindrome checker.
output-randomMachine_Keglovits.pdf	Output file for simple, random machine.

8. Programming Languages Used and Libraries: Python and pandas
9. Key data structures (for each sub-project): mostly used tuples, lists, dictionaries, and dataframes
10. General operation of code (for each subproject): Overall, the code operates very well. It first reads in the csv file and formats the data presented in a useable way, taking in the details of states, k-values, machine names, etc. as their own variables, and making the transition table into one overarching dictionary organized within itself. After this data is read in, the tapes are processed with the

machine, and a result is outputted, whether it be a rejection, acceptance, or halting due to a given halting condition.

11. What test cases you used/added, why you used them, what did they tell you about the correctness of your code: I was sure to include test cases that had different k-values, so the random Machine only has 2 tapes, as does the DNA complementer, but the palindrome machine has 4 tapes, and all machines have varying amounts of transitions and complexities within those transitions. Making sure to implement more complicated files, like the palindrome one, showed me places where I may have accidentally hard-coded numbers and they needed to be adjusted to be universally used, no matter the k-value, and they also showed me that it was important to know how to handle moments where the tape head moves past the length of the tape itself, because if it can't surpass the tape in some cases, it cannot come to an accept state.
12. To manage the code development, I broke it up into its component parts: reading in csv data, processing this data from a data frame to make it usable in the code, writing the machine, and handling its output.
13. Results: After a lot of trial and error, I found that my program works well as a k-tape DTM. The palindrome checker accurately accepts a given palindrome and rejects a non-palindrome, my simple machine hits an accept state in sequence and halts if the acceptance cannot be hit within a given condition of steps, and the DNA complementer creates a complementary tape of ATCG values from the given tape 1.
14. No team, just me.
15. What I'd do differently: honestly, not too much besides maybe starting earlier!
16. N/A