Project Readme Template

Version 1 9/11/24

A single copy of this template should be filled out and submitted with each project submission, regardless of the number of students on the team. It should have the name readme_"teamname"

Also change the title of this template to "Project x Readme Team xxx"

1	Team Name: dbrown39
2	Team members names and netids: Declan Brown - dbrown39
3	Overall project attempted, with sub-projects: (1) K-Tape Turing Machine
4	Overall success of the project: Very Successful
5	Approximately total time (in hours) to complete: 10 hrs
6	Link to github repository:https://github.com/dbrown39nd/TOCProject2
7	List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary). Add more rows as necessary.

File/folder Name	File Contents and Use
Code Files	
main_dbrown39.py src/tape_dbrown39.py src/TuringMachine_dbrown39.py	**main.py**: driver, reads all inputs from input/* and executes turing machine. **tape.py:** Tape class which is just a single tape object. Has methods: update_date, get_head, move_tape, and toString(). **TuringMachine_dbrown39.py**: K-tape Turing Machine that has methods execute, _check_input, _take_transition, _format_output,
Test Files	
input/test01_dbrown39.csv input/test02_dbrown39.csv input/test03_dbrown39.csv input/test04_dbrown39.csv	test01 and test02 are valid and invalid inputs, respectively, for the language w#w^R using 2 tapes, test03 and test04 are valid and invalid inputs, respectively, for the language w#w#w#w, using 4 tapes.

Output Files	
output/test01_dbrown39.txt output/test02_dbrown39.txt output/test03_dbrown39.txt output/test04_dbrown39.txt	test01 and test02 are the output trace of t1 and t2 respectively, for the language w#w^R using 2 tapes, test03 and test04 are the output traces of t2 and t3 respectively, for the language w#w#w#w, using 4 tapes.

8	Programming languages used, and associated libraries: I used python. I only used built in libraries like os, json, and csv.
9	Key data structures (for each sub-project): The transition function *delta* was a list[list[str]]. No other data structures used.
10	General operation of code (for each subproject) main.py -> parse input files, creating a list of Tape objects, and a TuringMachine object that takes in a list of tapes as one of the parameters, along with machine name, transition function *delta,* q_0, q_{accept} and q_{reject}. Then the main just runs TuringMachine.execute() which recursively takes transitions until accepts, rejects or reaches max iterations.

11 What test cases you used/added, why you used them, what did they tell you about the correctness of your code.

test01_dbrown39.csv

```
["Palindrome Checker: w#w^R", 2]
[1,0,1,#,1,0,1]
[1,0,1,#,1,0,1]
[q0, [1,1], q0, [1,1], [R,R]]
[q0, [0,0], q0, [0,0], [R,R]]
[q0, [#,#], q1, [#,#], [R,R]]
[q1, [1,1], q1, [1,1], [R,R]]
[q1, [0,0], q1, [0,0], [R,R]]
[q1, [_,_], qaccept, [_,_], [S,S]]
[q0, [1,0], qreject, [1,0], [S,S]]
[q0, [0,1], qreject, [0,1], [S,S]]
[q1, [1,0], qreject, [1,0], [S,S]]
[q1, [0,1], qreject, [0,1], [S,S]]
```

test03_dbrown39.csv:

```
["Four Copy Checker: w#w#w#w", 4]
[1,0,1,#,1,0,1,#,1,0,1,#,1,0,1]
[1,0,1,#,1,0,1,#,1,0,1,#,1,0,1]
[1,0,1,#,1,0,1,#,1,0,1,#,1,0,1]
```

```
[1,0,1,#,1,0,1,#,1,0,1,#,1,0,1]
[q0,  [1,1,1,1],  q0,  [1,1,1,1],  [R,R,R,R]]
[q0,  [0,0,0,0],  q0,  [0,0,0,0],  [R,R,R,R]]
[q0,  [#,#,#,#],  q1,  [#,#,#,#],  [R,R,R,R]]
[q1,  [1,1,1,1],  q1,  [1,1,1,1],  [R,R,R,R]]
[q1,  [0,0,0,0],  q1,  [0,0,0,0],  [R,R,R,R]]
[q1,  [#,#,#,#],  q2,  [#,#,#,#],  [R,R,R,R]]
[q2,  [1,1,1,1],  q2,  [1,1,1,1],  [R,R,R,R]]
[q2,  [0,0,0,0],  q2,  [0,0,0,0],  [R,R,R,R]]
[q2,  [#,#,#,#],  q3,  [#,#,#,#],  [R,R,R,R]]
[q3,  [1,1,1,1],  q3,  [1,1,1,1],  [R,R,R,R]]
[q3,  [0,0,0,0],  q3,  [0,0,0,0],  [R,R,R,R]]
[q3,  [_,_,_,_],  qaccept,  [_,_,_,_],  [S,S,S,S]]
[q0,  [1,1,1,0],  qreject,  [1,1,1,0],  [S,S,S,S]]
[q0,  [1,1,0,1],  qreject,  [1,1,0,1],  [S,S,S,S]]
[q0,  [1,0,1,1],  qreject,  [1,0,1,1],  [S,S,S,S]]
[q0,  [0,1,1,1],  qreject,  [0,1,1,1],  [S,S,S,S]]
[q1,  [1,1,1,0],  qreject,  [1,1,1,0],  [S,S,S,S]]
[q1,  [1,1,0,1],  qreject,  [1,1,0,1],  [S,S,S,S]]
[q1,  [1,0,1,1],  qreject,  [1,0,1,1],  [S,S,S,S]]
[q1,  [0,1,1,1],  qreject,  [0,1,1,1],  [S,S,S,S]]
[q2,  [1,1,1,0],  qreject,  [1,1,1,0],  [S,S,S,S]]
[q2,  [1,1,0,1],  qreject,  [1,1,0,1],  [S,S,S,S]]
[q2,  [1,0,1,1],  qreject,  [1,0,1,1],  [S,S,S,S]]
[q2,  [0,1,1,1],  qreject,  [0,1,1,1],  [S,S,S,S]]
[q3,  [1,1,1,0],  qreject,  [1,1,1,0],  [S,S,S,S]]
[q3,  [1,1,0,1],  qreject,  [1,1,0,1],  [S,S,S,S]]
[q3,  [1,0,1,1],  qreject,  [1,0,1,1],  [S,S,S,S]]
[q3,  [0,1,1,1],  qreject,  [0,1,1,1],  [S,S,S,S]]
```

I added two separate k-tape problems, one which checked if the tape was a palindrome using 2 tapes, and one which checked w#w#w#w, using 4 tapes. I chose these because they reflect the types of problems we have done all year, but this time with a turing machine and multiple tapes. Test cases test02 and test04 were also palindrome and 4 copy respectively, but their input tapes had a small adjustment to make them wrong, thus forcing a qreject. I wanted to ensure the machine was able to both accept and reject. These two cases worked well and proved that my code worked.

12 How you managed the code development

First I started with creating the **Tape** class, which was pretty simple. Then I created the **TuringMachine** class that had a list of tapes as an attribute. Then I created the test

	cases to test the TuringMachine. The implementation of the Tape and TuringMachine classes took me about 5-6 hours, and the testing took about 4 hours, because I had to go back and tweak my turing machine class many times. But breaking the problem down into two classes did simplify the development process.
13	Detailed discussion of results:

```
Machine: "Palindrome Checker w#w^R"
Number of Tapes: 2
Start State: q0
Accept State: qaccept
Reject State: qreject

+-------------------------T0-------------------------+
Current State: q0
T1: ['1', '0', '1', '#', '1', '0', '1']
       ^
T2: ['1', '0', '1', '#', '1', '0', '1']
       ^

+-------------------------T1-------------------------+
Current State: q0
T1: ['1', '0', '1', '#', '1', '0', '1']
            ^
T2: ['1', '0', '1', '#', '1', '0', '1']
            ^
Transition: ['q0', ['1', '1'], 'q0', ['1', '1'], ['R', 'R']]
New State: q0

+-------------------------T2-------------------------+
Current State: q0
T1: ['1', '0', '1', '#', '1', '0', '1']
                 ^
T2: ['1', '0', '1', '#', '1', '0', '1']
                 ^
Transition: ['q0', ['0', '0'], 'q0', ['0', '0'], ['R', 'R']]
New State: q0

+-------------------------T3-------------------------+
Current State: q0
T1: ['1', '0', '1', '#', '1', '0', '1']
                      ^
T2: ['1', '0', '1', '#', '1', '0', '1']
                      ^
Transition: ['q0', ['1', '1'], 'q0', ['1', '1'], ['R', 'R']]
New State: q0

+-------------------------T4-------------------------+
Current State: q1
T1: ['1', '0', '1', '#', '1', '0', '1']
                           ^
```

```
T2: ['1', '0', '1', '#', '1', '0', '1']
                     ^
Transition: ['q0', ['#', '#'], 'q1', ['#', '#'], ['R', 'R']]
New State: q1

+------------------------T5------------------------+
Current State: q1
T1: ['1', '0', '1', '#', '1', '0', '1']
                          ^
T2: ['1', '0', '1', '#', '1', '0', '1']
                          ^
Transition: ['q1', ['1', '1'], 'q1', ['1', '1'], ['R', 'R']]
New State: q1

+------------------------T6------------------------+
Current State: q1
T1: ['1', '0', '1', '#', '1', '0', '1']
                               ^
T2: ['1', '0', '1', '#', '1', '0', '1']
                               ^
Transition: ['q1', ['0', '0'], 'q1', ['0', '0'], ['R', 'R']]
New State: q1

+------------------------T7------------------------+
Current State: q1
T1: ['1', '0', '1', '#', '1', '0', '1', '_']
                                    ^
T2: ['1', '0', '1', '#', '1', '0', '1', '_']
                                    ^
Transition: ['q1', ['1', '1'], 'q1', ['1', '1'], ['R', 'R']]
New State: q1

+------------------------T8------------------------+
Current State: qaccept
T1: ['1', '0', '1', '#', '1', '0', '1', '_']
                                    ^
T2: ['1', '0', '1', '#', '1', '0', '1', '_']
                                    ^
Transition: ['q1', ['_', '_'], 'qaccept', ['_', '_'], ['S', 'S']]
New State: qaccept
Accept!
```

This was the output for test case 1 (input/test01_dbrown39.csv -> output/test01_dbrown39.txt)

The two-tape Turing machine checked if "101#101" was a valid palindrome by comparing each part of the string. The machine started reading both tapes from the left side, moving right while making sure the symbols on both tapes matched. First, it checked "101" on both tapes and found they were the same. When it got to the #

	symbol, it switched from q0 to q1, indicating we have passed halfway. After passing the #, it kept moving right and checking that the next "101" matched on both tapes too. When it got to the end and found blank spaces on both tapes, it knew it was done checking and said "yes, this is correct" by going to its accept mode. The machine would have said "no" if it found any symbols that didn't match, but since everything matched perfectly, it accepted the input as correct. It was cool to implement a turing machine that could use multiple tapes, and demonstrates how complex problems can be broken solved much easier with more tapes.
14	How the team was organized: I worked by myself.
15	What you might do differently if you did the project again Nothing, it was a pretty straightforward implementation of a k-tape tm.
16	Any additional material: