How the program is organized

The program is organized in the following modules:

- Data_Structure.py
- File_Handling.py
- Control.py
- DTMC_Calculations.py
- MarkovChain_TimeSeries.py
- ExperimentalStudy.py
- Main_DTMC.py

The first 6 modules contain code, structured as functions, which can solve the given tasks. In the last module (Main_DTMC.py), the program is tested and the functionality of the program is demonstrated. In the following, the most important functionality of the different parts is described:

Data_Structure.py

The data structure for a DTMC is a list containing the name of the DTMC, a dictionary of states and a dictionary of transitions. Transitions are lists containing the name of the transition, a source state, a target state and a probability. States only contain their given name.

A probability distribution is a list containing the name of the probability distribution, a reference to a DTMC it will be used on and a dictionary of state probabilities. The keys in the dictionary are the state names in referenced DTMC and the value is the probability of being in the given state.

We also have a data structure which is used to handle sets of DTMC's and probability distributions. This "container" is structured as a dictionary where the key is the name of a DTMC and the value is a list of probability distributions over this DTMC.

File_handling.py

This module contains all the code used to write DTMC's and probability distributions to files and to read these from a file and into a list of tokens. In addition there are functions which parse a list of tokens into the correct data structure, consisting of a container of DTMC's and probability distributions.

Control.py

This module contains multiple control functions used in a final function (`DTMCandProbDist_FinalCheckBeforeCalculations(probDist)`) which can check a probability distribution and the referenced DTMC for errors. This will be used before calculations to ensure that all logic is correct.

DTMC_Calculations.py

This code performs the mathematical operations needed to calculate the state probabilities after n iterations. For each iteration the code calculates new values for all the states in the DTMC. The new calculated state probability consists of two parts. The first is the equivalent state probability from the previous step multiplied with (1- the sum of all transition probabilities out of this state). In the second part the product of state probabilities (in the previous step) for the other states in the DTMC is multiplied with the transition probability into the given state. This product is added to the value from part 1 if the given state is a target state for these other states.

MarkovChain_TimeSeries.py
This module can generate a time series (a list of states) of length n from a DTMC. The code uses the random library to generate random numbers between 0 and 1. Each possible next state gets an interval based on the current state and transition probabilities and the intervals always add up to an interval [0,1).
Based on the random value, a new state is generated into the time series.

The code in this module can also generate a DTMC with transition probabilities based on a time series. The function counts the number of the different transitions from one type of state to another and based on these numbers it will calculate the different transition probabilities.

ExperimentalStudy.py
In this module there is code for performing the experimental study of comparing a DTMC and a new DTMC generated from a time series from the first DTMC. The code has functions that check if the states and transitions in the two markov chains are the same. The way it compares two DTMC's is that it calculates the mean absolute deviation between the transitions in the original DTMC and the DTMC generated from a time series of length n.

Main_DTMC.py
In this module, all the functionality of the program is tested. The benchmark from task 4 is tested. It takes in a txt-file with many different DTMC's and probability distributions. It breaks this file into a big list of tokens, creates instances of DTMC's and probability distributions and in the end, all legal instances are printed in a nice way into a new file called Results.txt.

In the main module we will also calculate the state probabilities after n steps given an initial probability distribution.

At the end of the program, an experimental study is performed. It will determine the necessary length of a time series, in order to get the transition probabilities close enough to the original DTMC. In the study we use two different DTMC's. The first is the DTMC "SeaCondition" given in the task consisting of 3 states and 4 transitions. The second one is "TrondheimWindCondition" consisting of 6 states and 10 transitions.

The results:

```
--Calculating state probabilities after 1000 steps--
State:               NOWIND      0.023076923076923033
State:          LIGHTBREEZE      0.1846153846153843
State:       MODERATEBREEZE      0.46153846153846073
State:                WINDY      0.11538461538461518
State:                STORM      0.15384615384615363
State:            HURRICANE      0.06153846153846146

----------------Experimental study----------------
DTMC: TrondheimWindCondition
Epsilon:             10 steps    None
Epsilon:            100 steps    0.11853489221775668
Epsilon:           1000 steps    0.036796998420221175
Epsilon:          10000 steps    0.024582926469549113
Epsilon:         100000 steps    0.02509884525934176


DTMC: SeaCondition
Epsilon:             10 steps    0.2
Epsilon:            100 steps    0.04738881401617252
Epsilon:           1000 steps    0.01808234122507913
Epsilon:          10000 steps    0.0012151068585634223
Epsilon:         100000 steps    0.0017144177594666105
```

In TrondheimWindCondition we can see that the difference between a time series of length 1000 and 10 000 is quite small but still big enough to make a difference. The difference between 10 000 and 100 000 steps is insignificant (0.0246 compared to 0.0251). Thus, for a Markov chain consisting of 6 states, it will not be necessary to generate more than 10 000 elements in the time series.

In SeaCondition, a DTMC with fewer states, we can see that a satisfactory solution is found faster. After the same amount of steps, a better solution is found compared to TrondheimWindCondition. After only 1000 steps the epsilon is smaller than the epsilon in TrondheimWindCondition after 100 000 steps. There is no significant difference in epsilon between 10 000 and 100 000 steps.