TASK:
Given the description of an NFA in the input json file, convert that machine into a DFA and write it's description to an output.json file.
PROCEDURE:
-> So, the code starts with importing the json library to bring in the input json and putting the data into an input object.
-> So, the code starts by identifying the various parts of the input json, and separating it into components.
//

```
with open('input.json') as json_file:
data = json.load(json_file)
# this gives me the input file just the way I want it
input_number_of_states = data['states'] # this just gave me the number of states
input_alphabet = data['letters']
input_function = data['t_func']
input_start = data['start']
input_final = data['final']
# we now have the parameters to start our work with, now we will construct the output dfa
output_number_of_states = pow(2, input_number_of_states)
outut_alphabet = input_alphabet
output_start = input_start
# output_function banana padhega , and output states bhi,
output_states = [] # this is just the list of output states
```

//
This is the initialisation code
-> Then we use the Bit Masking technique to generate all the possible output states. Along side this, we also generate the final states//

```
for i in range (0, pow(2, input_number_of_states)):
# now we will run that exponential thing
state = []
for j in range(0, input_number_of_states):
if (i & (1<<j)):
# then that means that weight in valid
state.append(j)
output_states.append(state)
for j in state:
if j in input_final:
output_final.append(state)
```

//
-> There after, we use a simple trick of converting the given input function to a dictionary which is of the form
"state" + "alphabet" : [Set of states it goes to]
This trick helps us later when we are forming the output transition function and we need to take unions of all the transitions
//

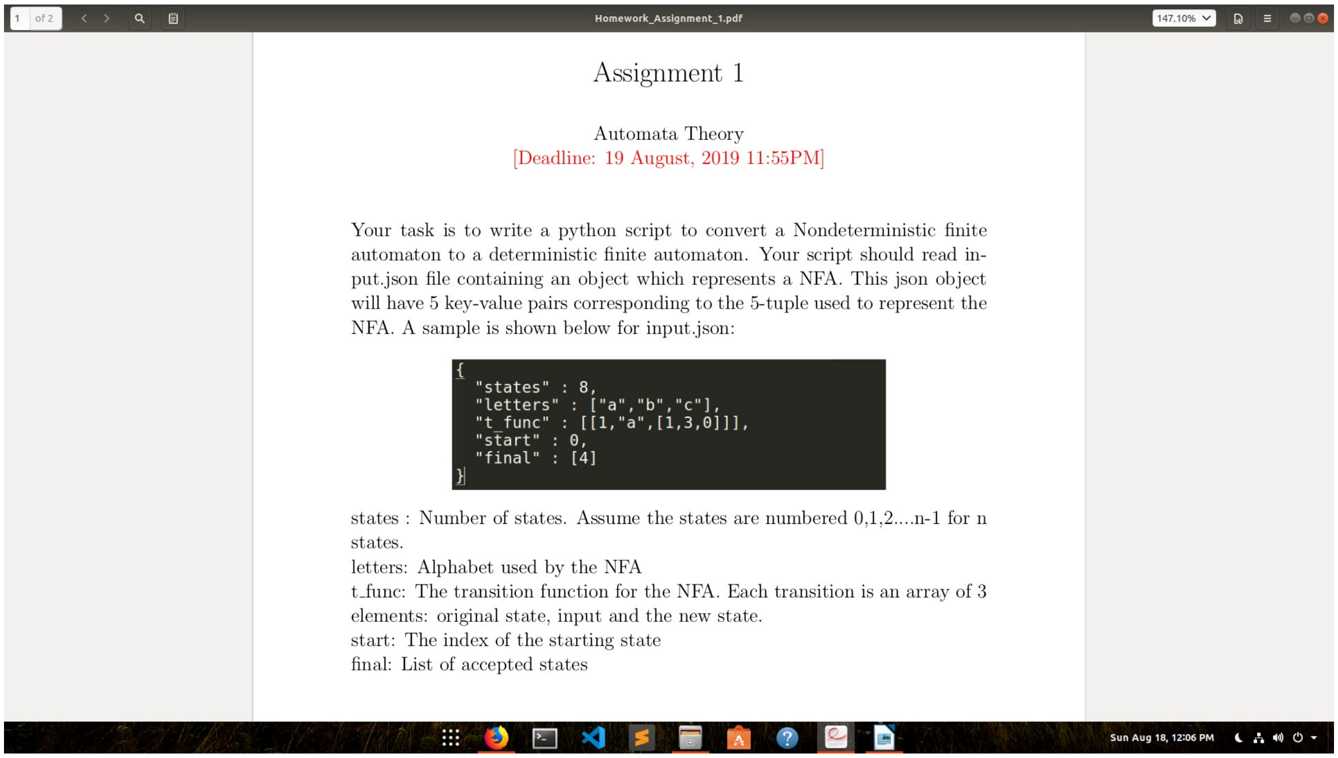
```
modified_input = {}
for ele in input_function:
a = str(ele[0])
b = ele[1]
```

```
c = a + b
modified_input[c] = ele[2]
# print(modified_input)
output_function = []
for state in output_states:
for letter in input_alphabet:
# I have to form a list that will have the current state
cur = []
cur.append(state)
cur.append(letter)codomain = []
# that gives me the elements, now I have to form the output
for s in state:
c = str(s) + str(letter)
if c in modified_input:
codomain.extend(modified_input[c])
set(codomain)
list(codomain)
cur.append(codomain)
output_function.append(cur)
for output in output_function:
print(output)
```

//

Then we do the necessary formalities for applying the finishing touches and wrap up the code by writing the output to an output.json file

INPUT FORMAT :



Assignment 1

Automata Theory
[Deadline: 19 August, 2019 11:55PM]

Your task is to write a python script to convert a Nondeterministic finite automaton to a deterministic finite automaton. Your script should read input.json file containing an object which represents a NFA. This json object will have 5 key-value pairs corresponding to the 5-tuple used to represent the NFA. A sample is shown below for input.json:

```
{
  "states" : 8,
  "letters" : ["a","b","c"],
  "t_func" : [[1,"a",[1,3,0]]],
  "start" : 0,
  "final" : [4]
}
```

states : Number of states. Assume the states are numbered 0,1,2....n-1 for n states.
letters: Alphabet used by the NFA
t_func: The transition function for the NFA. Each transition is an array of 3 elements: original state, input and the new state.
start: The index of the starting state
final: List of accepted states

OUTPUT FORMAT :

The Output follows the same format except for very few minor variations (eg. The parenthesis may be different etc.)

NOTES :

Please take care of the memory limits for the various data types used in the code. The only restrictions are those specified by the official python documentation