SRM INSTITUTE OF

SCIENCE &TECHNOLOGY

Kattankulathur

Chennai

|  |  |
| --- | --- |
| Name | * **GAURAV GUPTA** |
| Subject | * Advanced Programming practice |
| Section | * W2 |
| Roll No. | * RA2211026010284 |
| Title | * Assignment   week 12 |

**Assignment**

**Week 12**

**Q1) Write a Python program to create an NFA that accepts strings containing only the letter 'a'.**

**CODE:**

**from automata.fa.nfa import NFA**

**nfa=NFA(**

**states={'q0','q1'},**

**input\_symbols={'a'},**

**transitions={**

**'q0':{'a':{'q1'}},**

**'q1':{'a':{'q1'}}**

**},**

**initial\_state='q0',**

**final\_states={'q1'}**

**)**

**input\_string = input("Enter a string: ")**

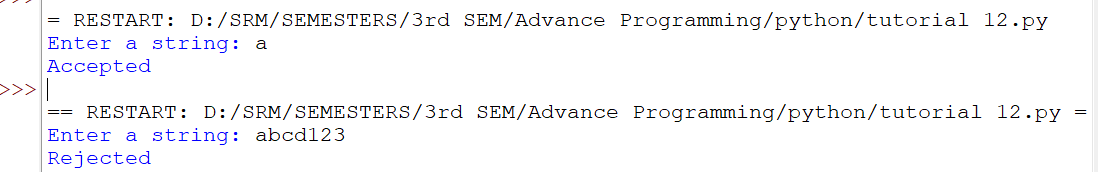
**if nfa.accepts\_input(input\_string):**

**print("Accepted")**

**else:**

**print("Rejected")**

**Output:**

****

**Q2) Create a Python function to check if a given string is accepted by an NFA that recognizes the pattern "ab|ba" (either "ab" or "ba").**

**CODE:**

**from automata.fa.nfa import NFA**

**nfa=NFA(**

**states={'q0','q1','q2'},**

**input\_symbols={'a','b'},**

**transitions={**

**'q0':{'a':{'q1'},'b':{'q2'}},**

**'q1':{'b':{'q2'}},**

**'q2':{'a':{'q1'}}**

**},**

**initial\_state='q0',**

**final\_states={'q1','q2'}**

**)**

**input\_string = input("Enter a string: ")**

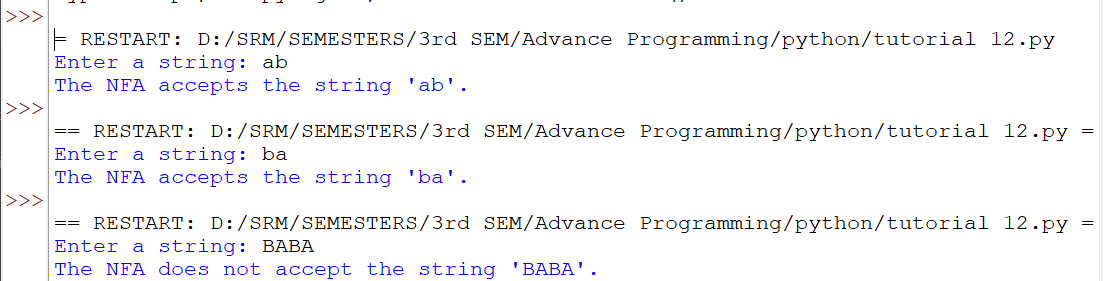
**if nfa.accepts\_input(input\_string):**

**print("Accepted")**

**else:**

**print("Rejected")**

**Output:**

****

**Q3) Implement a Python script that converts a simple NFA into a DFA with two states.**

**CODE:**

**# Define the NFA transitions**

**nfa\_transitions = {**

**('q0', 'a'): {'q1'},**

**('q1', 'b'): {'q2'},**

**('q2', 'a'): {'q1'},**

**('q0', ''): {'q2'}**

**}**

**# Initialize DFA variables**

**dfa\_states = set()**

**dfa\_transitions = {}**

**dfa\_initial = ('q0',)**

**dfa\_final = set()**

**# Initialize a queue for processing states**

**queue = [dfa\_initial]**

**while queue:**

**current\_states = queue.pop()**

**dfa\_states.add(current\_states)**

**for symbol in {'a', 'b'}:**

**next\_states = set()**

**for nfa\_state in current\_states:**

**epsilon\_transition = nfa\_transitions.get((nfa\_state, ''))**

**if epsilon\_transition:**

**next\_states.update(epsilon\_transition)**

**symbol\_transition = nfa\_transitions.get((nfa\_state, symbol))**

**if symbol\_transition:**

**next\_states.update(symbol\_transition)**

**next\_states = tuple(sorted(next\_states))**

**dfa\_transitions[(current\_states, symbol)] = next\_states**

**if next\_states not in dfa\_states:**

**queue.append(next\_states)**

**# Identify final states**

**for dfa\_state in dfa\_states:**

**for nfa\_state in dfa\_state:**

**if nfa\_state in {'q1', 'q2'}:**

**dfa\_final.add(dfa\_state)**

**# Print the DFA**

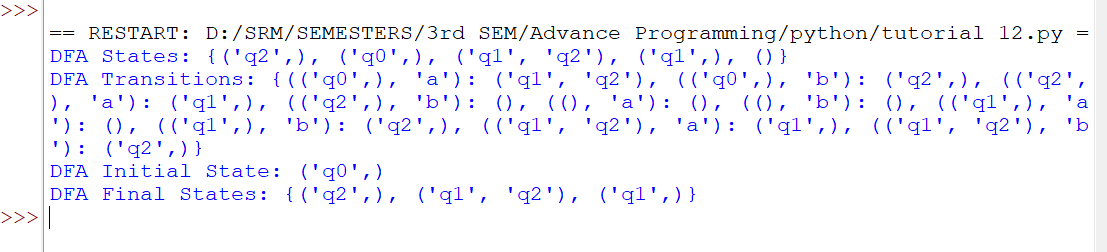
**print("DFA States:", dfa\_states)**

**print("DFA Transitions:", dfa\_transitions)**

**print("DFA Initial State:", dfa\_initial)**

**print("DFA Final States:", dfa\_final)**

**Output:**

****

**Q4) Write a Python program to construct a DFA that accepts binary strings ending in '01'.**

**CODE:**

**# Get the input binary string from the user**

**input\_string = input("Enter a binary string: ")**

**# Initialize current state**

**current\_state = 0**

**# Iterate through characters in the input string**

**for char in input\_string:**

**if current\_state == 0 and char == '0':**

**current\_state = 1**

**elif current\_state == 1 and char == '1':**

**current\_state = 2**

**else:**

**current\_state = 0**

**# Check if the last two characters are '01'**

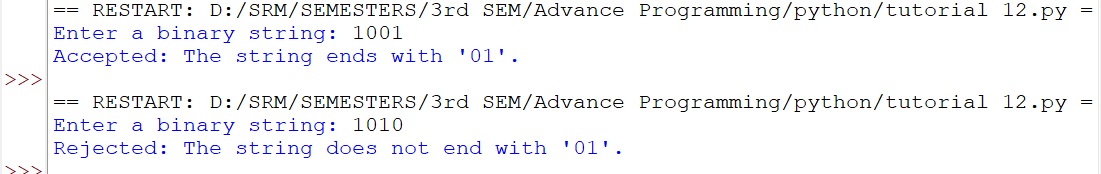
**if input\_string[-2:] == '01':**

**print("Accepted: The string ends with '01'.")**

**else:**

**print("Rejected: The string does not end with '01'.")**

**Output:**

****

**Q5) Develop a Python function that takes an NFA and returns the set of states that can be reached from a given state on a specific input symbol.**

**CODE:  
  
# Define the NFA properties**

**nfa = {**

**'states': {'q0', 'q1', 'q2'},**

**'alphabet': {'a', 'b'},**

**'transitions': {**

**'q0': {'a': {'q1'}, 'b': {'q2'}},**

**'q1': {'a': {'q1'}, 'b': {'q2'}},**

**'q2': {'a': {'q2'}, 'b': {'q2'}}**

**},**

**'initial\_state': 'q0',**

**'final\_states': {'q2'}**

**}**

**# Get input state and symbol from user**

**state = input("Enter a state: ")**

**symbol = input("Enter an input symbol: ")**

**# Find reachable states**

**reachable\_states = set()**

**for transition\_state, transitions in nfa['transitions'].items():**

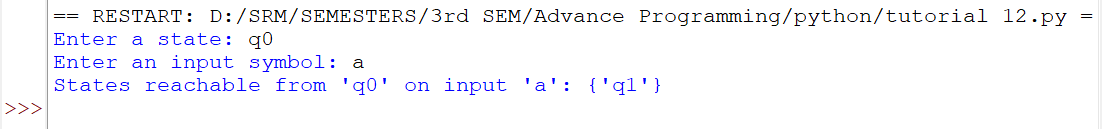
**if transition\_state == state and symbol in transitions:**

**reachable\_states.update(transitions[symbol])**

**# Print reachable states**

**print(f"States reachable from '{state}' on input '{symbol}':", reachable\_states)**

**Output:**

****

**Q6) Create a Python script to minimize a simple DFA with three states by merging equivalent states.**

**CODE:**

**# Define the DFA properties**

**dfa = {**

**'states': ['q0', 'q1', 'q2'],**

**'alphabet': {'0', '1'},**

**'transitions': {**

**'q0': {'0': 'q0', '1': 'q1'},**

**'q1': {'0': 'q2', '1': 'q1'},**

**'q2': {'0': 'q0', '1': 'q1'},**

**},**

**'initial\_state': 'q0',**

**'final\_states': ['q2']**

**}**

**# Step 1: Split states into two sets - final states and non-final states**

**final\_states = set(dfa['final\_states'])**

**non\_final\_states = set(dfa['states']) - final\_states**

**# Step 2: Initialize partition with the two sets**

**partition = [final\_states, non\_final\_states]**

**# Step 3: Refine the partition using the transitions**

**while True:**

**new\_partition = []**

**for group in partition:**

**new\_groups = []**

**for symbol in dfa['alphabet']:**

**target\_states = set()**

**for state in group:**

**target\_state = dfa['transitions'][state][symbol]**

**target\_states.add(target\_state)**

**for subgroup in new\_groups:**

**if target\_states == subgroup:**

**subgroup.update(group)**

**break**

**else:**

**new\_groups.append(target\_states)**

**if len(new\_groups) > 1:**

**break**

**else:**

**new\_groups = [group]**

**new\_partition.extend(new\_groups)**

**if new\_partition == partition:**

**break**

**partition = new\_partition**

**# Step 4: Build the new DFA**

**new\_states = []**

**new\_transitions = {}**

**for group in partition:**

**new\_state = ','.join(sorted(group))**

**new\_states.append(new\_state)**

**for state in group:**

**if state in final\_states:**

**final\_state = new\_state**

**break**

**else:**

**final\_state = None**

**new\_transitions[new\_state] = {}**

**for symbol in dfa['alphabet']:**

**target\_states = set()**

**for state in group:**

**target\_state = dfa['transitions'][state][symbol]**

**target\_states.add(target\_state)**

**for subgroup in partition:**

**if target\_states == subgroup:**

**new\_transitions[new\_state][symbol] = ','.join(sorted(subgroup))**

**break**

**minimized\_dfa = {**

**'states': new\_states,**

**'alphabet': dfa['alphabet'],**

**'transitions': new\_transitions,**

**'initial\_state': ','.join(sorted(partition[0])),**

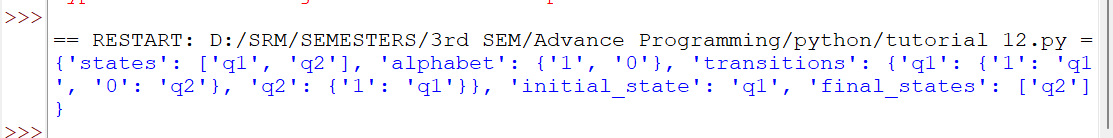
**'final\_states': [final\_state] if final\_state is not None else [],**

**}**

**# Print the minimized DFA**

**print(minimized\_dfa)**

**Output:**

****

**Q7) Implement a Python function that checks if a given string is accepted by a DFA that recognizes the pattern "ab\*c".**

**CODE:**

**# Define the DFA properties**

**dfa = {**

**'states': {'q0', 'q1', 'q2', 'q3'},**

**'alphabet': {'a', 'b', 'c'},**

**'transitions': {**

**'q0': {'a': 'q1'},**

**'q1': {'b': 'q2'},**

**'q2': {'b': 'q2', 'c': 'q3'},**

**'q3': {}**

**},**

**'initial\_state': 'q0',**

**'final\_states': {'q3'}**

**}**

**# Get input string from user**

**input\_string = input("Enter a string: ")**

**# Initialize current state**

**current\_state = dfa['initial\_state']**

**# Iterate through characters in the input string**

**for symbol in input\_string:**

**if symbol not in dfa['alphabet']:**

**print(f"Invalid symbol '{symbol}' in the input.")**

**break**

**current\_state = dfa['transitions'][current\_state].get(symbol)**

**if current\_state is None:**

**break**

**# Check if the current state is a final state**

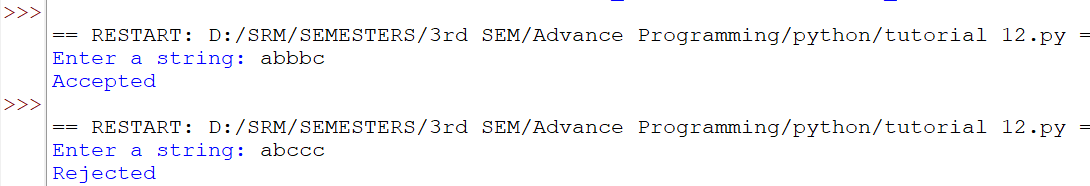
**if current\_state in dfa['final\_states']:**

**print("Accepted")**

**else:**

**print("Rejected")**

**Output:**

****

**Q8) Write a Python program to create an NFA that accepts strings with an odd number of '1's.**

**CODE:**

**# Define the NFA properties**

**nfa = {**

**'states': {'q0', 'q1'},**

**'input\_symbols': {'0', '1'},**

**'transitions': {**

**'q0': {'0': {'q0'}, '1': {'q0', 'q1'}},**

**'q1': {'0': {'q1'}, '1': {'q1'}}**

**},**

**'initial\_state': 'q0',**

**'final\_states': {'q1'}**

**}**

**# Get input string from user**

**input\_string = input("Enter a string: ")**

**# Initialize current states as a set with the initial state**

**current\_states = {'q0'}**

**# Iterate through characters in the input string**

**for symbol in input\_string:**

**if symbol not in nfa['input\_symbols']:**

**print(f"Invalid symbol '{symbol}' in the input.")**

**break**

**next\_states = set()**

**for state in current\_states:**

**next\_states.update(nfa['transitions'][state].get(symbol, set()))**

**current\_states = next\_states**

**# Check if the final state is in the set of current states and the input string has an odd number of '1's**

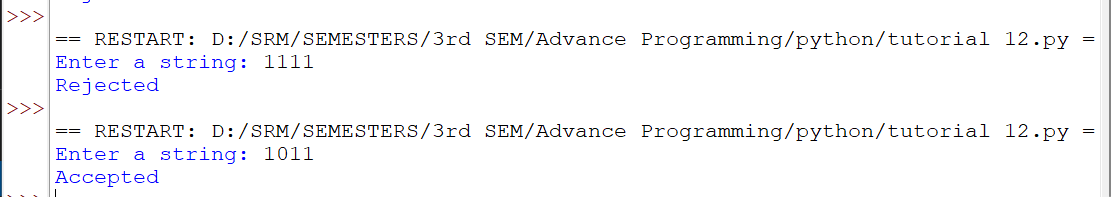
**if 'q1' in current\_states and input\_string.count('1') % 2 != 0:**

**print("Accepted")**

**else:**

**print("Rejected")**

**Output:**

****

**Q9) Develop a Python function that converts a simple regular expression like "a(b|c)\*" into an equivalent NFA.**

**CODE:**

**# Define the NFA properties**

**nfa = {**

**'states': {'q0', 'q1', 'q2', 'q3'},**

**'input\_symbols': {'a', 'b', 'c'},**

**'transitions': {**

**'q0': {'a': {'q1'}},**

**'q1': {'b': {'q2', 'q1'}, 'c': {'q2', 'q1'}},**

**'q2': {'b': {'q2'}, 'c': {'q2'}},**

**'q3': {}**

**},**

**'initial\_state': 'q0',**

**'final\_states': {'q2'}**

**}**

**# Get input string from user**

**input\_string = input("Enter a string: ")**

**# Initialize current states as a set with the initial state**

**current\_states = {'q0'}**

**# Iterate through characters in the input string**

**for symbol in input\_string:**

**if symbol not in nfa['input\_symbols']:**

**print(f"Invalid symbol '{symbol}' in the input.")**

**break**

**next\_states = set()**

**for state in current\_states:**

**next\_states.update(nfa['transitions'][state].get(symbol, set()))**

**current\_states = next\_states**

**# Check if the final state is in the set of current states**

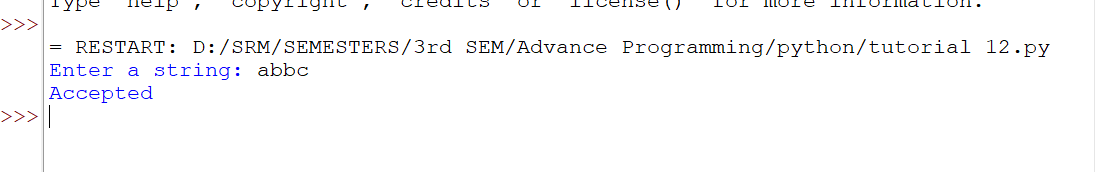
**if 'q2' in current\_states:**

**print("Accepted")**

**else:**

**print("Rejected")**

**Output:**

****