1.Write a deterministic automata code for the language L(M)={w|wε{0,1}*}and W is a string that does not contain consecutive 0's.

```
class DFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2'}
        self.alphabet = {'0', '1'}
        self.transitions = {
            ('q0', '0'): 'q1',
            ('q0', '1'): 'q0',
            ('q1', '0'): 'q2',
            ('q1', '1'): 'q0',
            ('q2', '0'): 'q2',
            ('q2', '1'): 'q0',
        }
        self.start_state = 'q0'
        self.accept_states = {'q0', 'q1', 'q2'}

    def run(self, input_string):
        current_state = self.start_state
        for symbol in input_string:
            current_state = self.transitions[(current_state, symbol)]
        return current_state in self.accept_states
dfa = DFA()
input_strings = ['0', '1', '01', '10', '11', '100', '101', '110']
for input_string in input_strings:
    if dfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
0 is accepted
1 is accepted
01 is accepted
10 is accepted
11 is accepted
100 is accepted
101 is accepted
110 is accepted
```

2.Write a deterministic automata code for the language with ∑={0,1}accepts the set of all strings with three consecutive 1's. class

```
class DFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2', 'q3'}
        self.alphabet = {'0', '1'}
        self.transitions = {
            ('q0', '0'): 'q0',
            ('q0', '1'): 'q1',
            ('q1', '0'): 'q0',
            ('q1', '1'): 'q2',
            ('q2', '0'): 'q0',
            ('q2', '1'): 'q3',
            ('q3', '0'): 'q0',
            ('q3', '1'): 'q3',
        }
        self.start_state = 'q0'
        self.accept_states = {'q3'}

    def run(self, input_string):
        current_state = self.start_state
        for symbol in input_string:
            current_state = self.transitions[(current_state, symbol)]
        return current_state in self.accept_states
dfa = DFA()
input_strings = ['0', '1', '01', '10', '11', '111', '0111', '10111', '1110']
for input_string in input_strings:
    if dfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
0 is rejected
1 is rejected
01 is rejected
10 is rejected
11 is rejected
111 is accepted
0111 is accepted
10111 is accepted
1110 is rejected
```

3.Write a deterministic automata code for the language with ∑={0,1}accepts even number of 0's and even number of 1's.

```
class DFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2', 'q3'}
        self.alphabet = {'0', '1'}
        self.transitions = {
            ('q0', '0'): 'q1',
            ('q0', '1'): 'q0',
            ('q1', '0'): 'q0',
            ('q1', '1'): 'q2',
            ('q2', '0'): 'q3',
            ('q2', '1'): 'q1',
            ('q3', '0'): 'q2',
            ('q3', '1'): 'q3',
        }
        self.start_state = 'q0'
        self.accept_states = {'q0'}

    def run(self, input_string):
        current_state = self.start_state
        for symbol in input_string:
            current_state = self.transitions[(current_state, symbol)]
        return current_state in self.accept_states
dfa = DFA()
input_strings = ['0', '1', '01', '10', '11', '0000', '0001', '0011', '0101', '01
for input_string in input_strings:
    if dfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
0 is rejected
1 is accepted
01 is rejected
10 is rejected
11 is accepted
0000 is accepted
0001 is rejected
0011 is accepted
0101 is rejected
0111 is rejected
```

4.Write a deterministic automata code for the language with ∑={0,1}accepts the only input 101

```
class DFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2'}
        self.alphabet = {'0', '1'}
        self.transitions = {
            ('q0', '1'): 'q1',
            ('q1', '0'): 'q2',
            ('q2', '1'): 'q2',
            ('q2', '0'): 'q2'
        }
        self.start_state = 'q0'
        self.accept_states = {'q2'}

    def run(self, input_string):
        current_state = self.start_state
        for symbol in input_string:
            current_state = self.transitions.get((current_state, symbol), None)
            if current_state is None:
                return False
        return current_state in self.accept_states
dfa = DFA()
input_strings = ['101', '1', '10', '100', '1010', '0101']
for input_string in input_strings:
    if dfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
101 is accepted
1 is rejected
10 is accepted
100 is accepted
1010 is accepted
0101 is rejected
```

5.Write a deterministic automata code for the language with ∑={0,1}accepts those string which starts with 1 and ends with 0.

```
class DFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2'}
        self.alphabet = {'0', '1'}
        self.transitions = {
            ('q0', '1'): 'q1',
            ('q1', '0'): 'q2',
            ('q1', '1'): 'q1',
            ('q2', '0'): 'q2',
            ('q2', '1'): 'q1'
        }
        self.start_state = 'q0'
        self.accept_states = {'q2'}

    def run(self, input_string):
        current_state = self.start_state
        for symbol in input_string:
            current_state = self.transitions.get((current_state, symbol), None)
            if current_state is None:
                return False
        return current_state in self.accept_states
dfa = DFA()
input_strings = ['10', '110', '1010', '100', '1110', '1']
for input_string in input_strings:
    if dfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
10 is accepted
110 is accepted
1010 is accepted
100 is accepted
1110 is accepted
1 is rejected
```

6.Give a non-deterministic automata code for (a|b)*aab.

```
class NFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2', 'q3'}
        self.alphabet = {'a', 'b'}
        self.transitions = {
            ('q0', 'a'): {'q0', 'q1'},
            ('q0', 'b'): {'q0', 'q3'},
            ('q1', 'a'): {'q2'},
            ('q2', 'b'): {'q3'}
        }
        self.start_state = 'q0'
        self.accept_states = {'q3'}

    def run(self, input_string):
        current_states = {self.start_state}
        for symbol in input_string:
            next_states = set()
            for state in current_states:
                next_states |= self.transitions.get((state, symbol), set())
            current_states = next_states
        return bool(current_states & self.accept_states)
nfa = NFA()
input_strings = ['aab', 'baabaab', 'a', 'b', 'aa', 'ab']
for input_string in input_strings:
    if nfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
aab is accepted
baabaab is accepted
a is rejected
b is accepted
aa is rejected
ab is accepted
```

7.Give a non-deterministic automata code for the set of all binary strings that have either the number of 0's odd,or the number of 1's not a multiple of 3,or both.

```
class NFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7'}
        self.alphabet = {'0', '1'}
        self.transitions = {
            ('q0', '0'): {'q1'},
            ('q0', '1'): {'q2'},
```

```
            ('q1', '0'): {'q0'},
            ('q1', '1'): {'q3'},
            ('q2', '0'): {'q4'},
            ('q2', '1'): {'q0'},
            ('q3', '0'): {'q5'},
            ('q3', '1'): {'q1'},
            ('q4', '0'): {'q2'},
            ('q4', '1'): {'q6'},
            ('q5', '0'): {'q7'},
            ('q5', '1'): {'q3'},
            ('q6', '0'): {'q4'},
            ('q6', '1'): {'q7'},
            ('q7', '0'): {'q5'},
            ('q7', '1'): {'q6'},
        }
        self.start_state = 'q0'
        self.accept_states = {'q1', 'q3', 'q4', 'q5', 'q6', 'q7'}

    def run(self, input_string):
        current_states = {self.start_state}
        for symbol in input_string:
            next_states = set()
            for state in current_states:
                next_states |= self.transitions.get((state, symbol), set())
            current_states = next_states
        return bool(current_states & self.accept_states)
nfa = NFA()
input_strings = ['0', '1', '00', '01', '10', '11', '000', '001', '010', '011', '
for input_string in input_strings:
    if nfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
0 is accepted
1 is rejected
00 is rejected
01 is accepted
10 is accepted
11 is rejected
000 is accepted
001 is rejected
010 is accepted
011 is accepted
100 is rejected
101 is accepted
110 is accepted
111 is rejected
```

8.Give a non-deterministic automata code for the language L=(ab)(ba)Uaa*.

```
class NFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7'}
        self.alphabet = {'a', 'b'}
        self.transitions = {
            ('q0', 'a'): {'q1', 'q7'},
            ('q0', 'b'): {'q0', 'q4'},
            ('q1', 'a'): {'q2'},
            ('q1', 'b'): {'q1', 'q5'},
            ('q2', 'a'): {'q2'},
            ('q2', 'b'): {'q3'},
            ('q3', 'a'): {'q7'},
            ('q3', 'b'): {'q0'},
            ('q4', 'a'): {'q7'},
            ('q4', 'b'): {'q4', 'q6'},
            ('q5', 'a'): {'q1', 'q7'},
            ('q5', 'b'): {'q5'},
            ('q6', 'a'): {'q7'},
            ('q6', 'b'): {'q4'},
            ('q7', 'a'): {'q7'},
            ('q7', 'b'): {'q7'},
        }
        self.start_state = 'q0'
        self.accept_states = {'q2', 'q3', 'q5', 'q7'}

    def run(self, input_string):
        current_states = {self.start_state}
        for symbol in input_string:
            next_states = set()
            for state in current_states:
                next_states |= self.transitions.get((state, symbol), set())
            current_states = next_states
        return bool(current_states & self.accept_states)
nfa = NFA()
input_strings = ['ab', 'ba', 'abba', 'baab', 'aba', 'baa', 'aab', 'aa', 'bbb', '
for input_string in input_strings:
    if nfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
ab is accepted
ba is accepted
abba is accepted
baab is accepted
aba is accepted
baa is accepted
aab is accepted
aa is accepted
bbb is rejected
b is rejected
a is accepted
bb is rejected
bbaa is accepted
abaa is accepted
aabaa is accepted
```

9.Give a non-deterministic automata code for the language L that have at least two consecutive 0's or 1's.

```
class NFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2', 'q3', 'q4'}
        self.alphabet = {'0', '1'}
        self.transitions = {
            ('q0', '0'): {'q1', 'q0'},
            ('q0', '1'): {'q2', 'q0'},
            ('q1', '0'): {'q3'},
            ('q1', '1'): {'q2'},
            ('q2', '0'): {'q1'},
            ('q2', '1'): {'q4'},
            ('q3', '0'): {'q3'},
            ('q3', '1'): {'q3'},
            ('q4', '0'): {'q4'},
            ('q4', '1'): {'q4'},
        }
        self.start_state = 'q0'
        self.accept_states = {'q1', 'q2', 'q3', 'q4'}

    def run(self, input_string):
        current_states = {self.start_state}
        for symbol in input_string:
            next_states = set()

            for state in current_states:
                next_states |= self.transitions.get((state, symbol), set())
            current_states = next_states
        return bool(current_states & self.accept_states)
```

```
nfa = NFA()
input_strings = ['0', '1', '00', '11', '01', '10', '000', '111', '010', '101', '
for input_string in input_strings:
    if nfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
0 is accepted
1 is accepted
00 is accepted
11 is accepted
01 is accepted
10 is accepted
000 is accepted
111 is accepted
010 is accepted
101 is accepted
001 is accepted
100 is accepted
011 is accepted
110 is accepted
```

10.Give a non-deterministic automata code for the language L=(01U010)*.

```
class NFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2', 'q3', 'q4'}
        self.alphabet = {'0', '1'}
        self.transitions = {
            ('q0', '0'): {'q1'},
            ('q0', '1'): {'q0'},
            ('q1', '0'): {'q2'},
            ('q1', '1'): {'q0'},
            ('q2', '0'): {'q1', 'q3'},
            ('q2', '1'): {'q0'},
            ('q3', '0'): {'q1'},
            ('q3', '1'): {'q4'},
            ('q4', '0'): {'q1'},
            ('q4', '1'): {'q0'},
        }
        self.start_state = 'q0'
        self.accept_states = {'q0'}

    def run(self, input_string):
        current_states = {self.start_state}
        for symbol in input_string:
            next_states = set()
            for state in current_states:
                next_states |= self.transitions.get((state, symbol), set())
            current_states = next_states
        return bool(current_states & self.accept_states)
nfa = NFA()
input_strings = ['', '01', '010', '01001', '010010', '01010101', '010011010']
for input_string in input_strings:
    if nfa.run(input_string):
        print(f'{input_string} is accepted')
    else:
        print(f'{input_string} is rejected')
```

```
 is accepted
01 is accepted
010 is rejected
01001 is accepted
010010 is rejected
01010101 is accepted
010011010 is rejected
```

Colab paid products - Cancel contracts here

✓ 0s completed at 9:59 PM