APP WEEK 13

Sample programs to Construct NFA and DFA using Python

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

q1.py > ...

```python
def is_accepting_state(current_state):
    return current_state == 1

def nfa_accepts(input_string):
    current_state = 0
    for char in input_string:
        if current_state == 0 and char == 'a':
            current_state = 1
        elif current_state == 1 and char == 'a':
            current_state = 1

    return is_accepting_state(current_state)

# Get the input string from the user
input_string = input("Enter a string to check if it contains only 'a's: ")

if nfa_accepts(input_string):
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
PS C:\Users\MANI\Desktop\week 13> & C:/Python312/python.exe "c:/Users/MANI/Desktop/week 13/q1.py"
Could not find platform independent libraries <prefix>
Enter a string to check if it contains only 'a's: belgium
Rejected: The string does not contain only 'a's.
```

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").

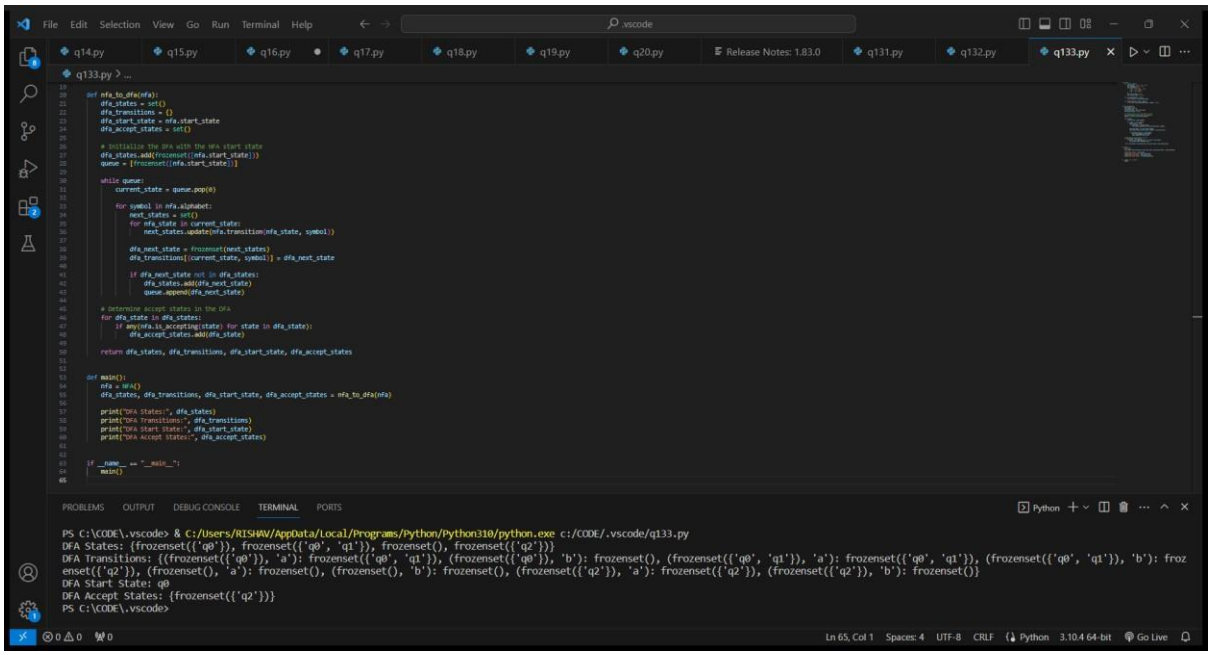
Q2.PY > ...

```python
def is_accepting_state(current_state):
    return current_state in [2, 4]

def nfa_accepts(input_string):
    current_state = 0

    for char in input_string:
        if current_state == 0 and char == 'a':
            current_state = 1
        elif current_state == 0 and char == 'b':
            current_state = 3
        elif current_state == 1 and char == 'b':
            current_state = 2
        elif current_state == 2:
            break  # Accept if "ab"
        elif current_state == 3 and char == 'a':
            current_state = 4
```

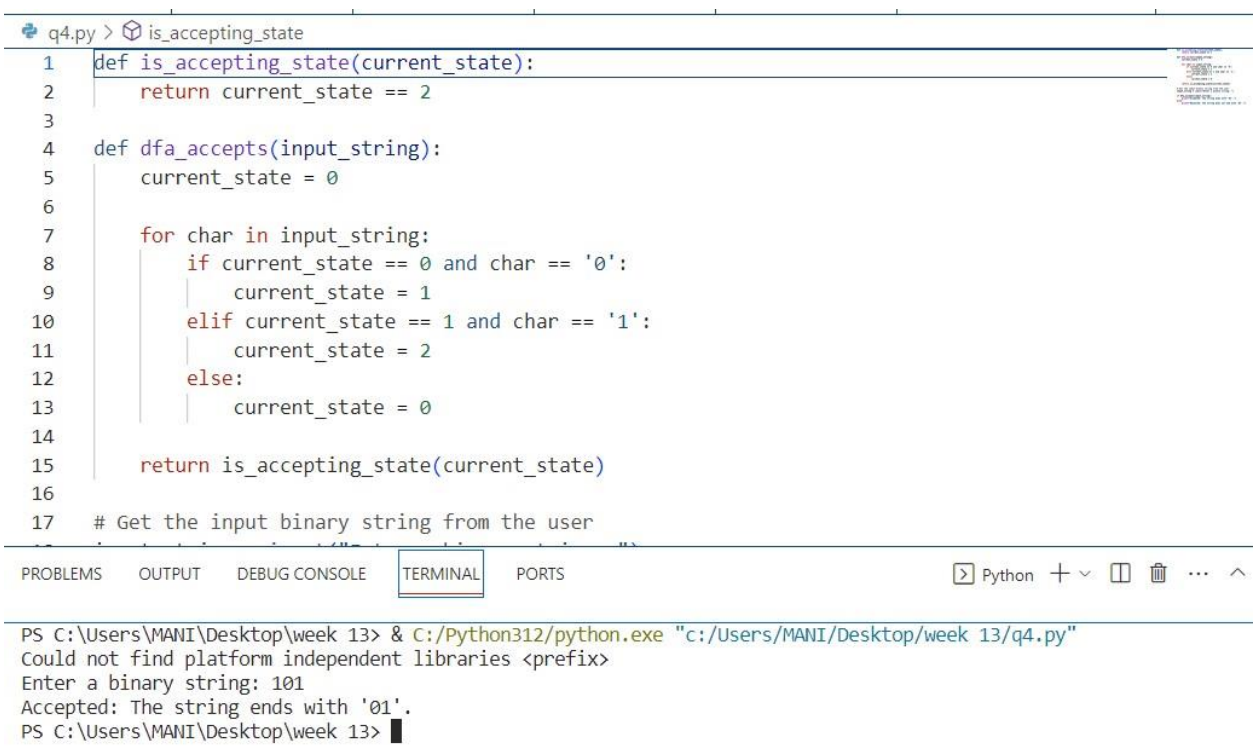
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
PS C:\Users\MANI\Desktop\week 13> & C:/Python312/python.exe "c:/Users/MANI/Desktop/week 13/Q2.PY"
Could not find platform independent libraries <prefix>
Enter a string to check if it matches 'ab|ba': bangalore
Accepted: The string matches 'ab|ba'.
```

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



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



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.

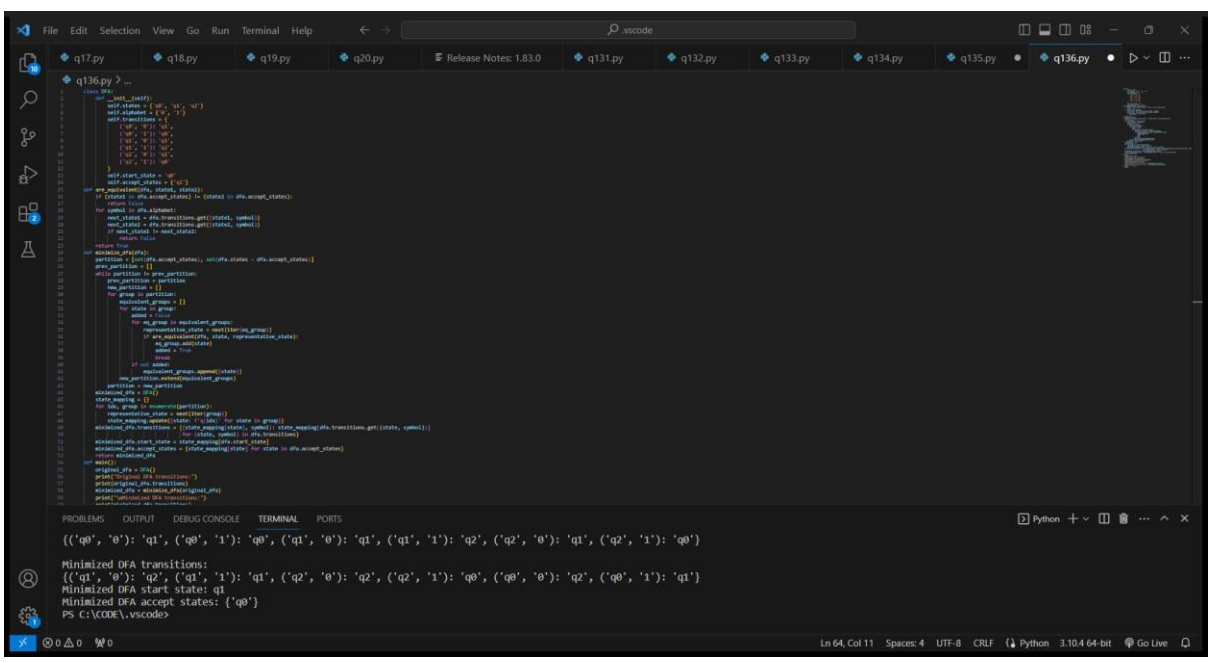
q5.py > ...

```python
def nfa_transition(nfa, current_states, symbol):
    next_states = set()

    for state in current_states:
        if state in nfa and symbol in nfa[state]:
            next_states |= nfa[state][symbol]

    return next_states

# Example NFA
nfa = {
    0: {'a': {0, 1}},
    1: {'b': {2}},
    2: {'c': {3}},
    3: {'d': {4}},
    4: {'e': {5}},
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
PS C:\Users\MANI\Desktop\week 13> & C:/Python312/python.exe "c:/Users/MANI/Desktop/week 13/q5.py"
Could not find platform independent libraries <prefix>
From state 0 on symbol 'a', we can reach states: {0, 1}
```

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



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

```python
class DFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2'}
        self.alphabet = {'a', 'b', 'c'}
        self.transitions = {
            ('q0', 'a'): 'q1',
            ('q1', 'b'): 'q1',
            ('q1', 'c'): 'q2',
            ('q2', 'b'): 'q1'
        }
        self.start_state = 'q0'
        self.accept_state = 'q2'
    def transition(self, state, symbol):
        return self.transitions.get((state, symbol), None)
def is_accepted_by_dfa(input_str):
    dfa = DFA()
    current_state = dfa.start_state
    for symbol in input_str:
        next_state = dfa.transition(current_state, symbol)
        if next_state is None:
            return False
        current_state = next_state
    return current_state == dfa.accept_state
def main():
    dfa = DFA()
    test_strings = ["abbc", "ac", "abc", "abbbc", "aabbc", "ab"]
    for string in test_strings:
        if is_accepted_by_dfa(string):
            print(f"'{string}' is accepted")
        else:
            print(f"'{string}' is not accepted")
if __name__ == "__main__":
    main()
```

```
PS C:\CODE\.vscode> & C:/Users/RISHAV/AppData/Local/Programs/Python/Python310/python.exe c:/CODE/.vscode/q137.py
'abbc' is accepted
'ac' is accepted
'abc' is accepted
'abbbc' is accepted
'aabbc' is not accepted
'ab' is not accepted
PS C:\CODE\.vscode>
```

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

```python
class NFA:
    def __init__(self):
        self.states = {'q0', 'q1', 'q2'}
        self.alphabet = {'0', '1'}
        self.transitions = {
            ('q0', '0'): {'q0'},
            ('q0', '1'): {'q1'},
            ('q1', '0'): {'q0'},
            ('q1', '1'): {'q2'},
            ('q2', '0'): {'q2'},
            ('q2', '1'): {'q1'}
        }
        self.start_state = 'q0'
        self.accept_states = {'q1'}
    def is_accepting(self, state):
        return state in self.accept_states
    def transition(self, state, symbol):
        return self.transitions.get((state, symbol), set())
    def accept(self, input_str):
        current_states = {self.start_state}
        for symbol in input_str:
            next_states = set()
            for state in current_states:
                next_states.update(self.transition(state, symbol))
            current_states = next_states
        return any(self.is_accepting(state) for state in current_states)
def main():
    nfa = NFA()
    test_strings = ["", "0", "1", "11", "101", "11011"]
    for string in test_strings:
        if nfa.accept(string):
            print(f"'{string}' is accepted (odd number of '1's)")
        else:
            print(f"'{string}' is not accepted (even number of '1's)")
if __name__ == "__main__":
    main()
```

```
PS C:\CODE\.vscode> & C:/Users/RISHAV/AppData/Local/Programs/Python/Python310/python.exe c:/CODE/.vscode/q138.py
'' is not accepted (even number of '1's)
'0' is not accepted (even number of '1's)
'1' is accepted (odd number of '1's)
'11' is not accepted (even number of '1's)
'101' is accepted (odd number of '1's)
'11011' is not accepted (even number of '1's)
```

Q9) Write a Python program to create an NFA that accepts strings with an odd number of '1's.