ToC Practical:

Practical 1:. Design a Program for creating machine that accepts three consecutive one.

Code:

```
states = {
    "q0": {"0": "q0", "1": "q1"},
    "q1": {"0": "q0", "1": "q2"},
    "q2": {"0": "q0", "1": "q3"},
    "q3": {"0": "q3", "1": "q3"}
}

initial_state = "q0"
final_state = "q3"

def check_string_recursive(string: str, current_state: str):
    if not string:
        return current_state == final_state
    next_state = states[current_state].get(string[0])
    return next_state and check_string_recursive(string[1:], next_state)

user_input = input("Enter a binary string: ")
print("Accepted" if set(user_input) <= {"0", "1"} and check_string_recursive(user_input,
initial_state) else "Not accepted")
```

Practical 2:. Design a Program for creating machine that accepts the string always ending with 101.

Code:

```
states = {
    "q0": {"0": "q0", "1": "q1"},
    "q1": {"0": "q2", "1": "q1"},
    "q2": {"0": "q0", "1": "q3"},
    "q3": {"0": "q2", "1": "q1"}
}

initial_state = "q0"
final_state = "q3"

def check_string_recursive(string: str, current_state: str):
    if not string:
        return current_state == final_state
    next_state = states[current_state].get(string[0])
```

```
    return next_state and check_string_recursive(string[1:], next_state)

user_input = input("Enter a binary string: ")
print("Accepted" if set(user_input) <= {"0", "1"} and check_string_recursive(user_input,
initial_state) else "Not accepted")
```

Practical 3:. Design a program for accepting decimal number divisible by 2.

Code:

```
class DFA:
    def __init__(self):
        self.state = "q0"

    def transition(self, char):
        if self.state == "q0":
            self.state = "q0" if char == '0' else "q1"
        elif self.state == "q1":
            self.state = "q0" if char == '0' else "q1"

    def is_accepted(self, binary_string):
        for char in binary_string:
            self.transition(char)
        return self.state == "q0"

def check_divisibility_by_2(decimal_number):
    binary_number = bin(decimal_number)[2:]
    print(f"Binary representation: {binary_number}")
    dfa = DFA()
    if dfa.is_accepted(binary_number):
        print("Accepted: The binary number is divisible by 2.")
    else:
        print("Rejected: The binary number is not divisible by 2.")

decimal_number = int(input("Enter a decimal number: "))
check_divisibility_by_2(decimal_number)
```

Practical 6:. Write a program for generating derivation sequence / language for the given sequence of productions.

Code:

```
def generate_derivation(productions, start_symbol, target_string):
    queue = [(start_symbol, [start_symbol])]

    while queue:
        current_string, derivation = queue.pop(0)

        if current_string == target_string:
            return derivation

        for lhs in productions:
            if lhs in current_string:
                for replacement in productions[lhs]:
                    new_string = current_string.replace(lhs, replacement, 1)
                    new_derivation = derivation + [new_string]
                    queue.append((new_string, new_derivation))

    return None

productions = {}
num_rules = int(input("Enter number of production rules: "))

for _ in range(num_rules):
    lhs, rhs = input("Enter production rule (e.g., S -> aA | bB): ").split("->")
    lhs = lhs.strip()
    rhs = [x.strip() for x in rhs.split("|")]
    productions[lhs] = rhs

start_symbol = input("Enter start symbol: ").strip()
target_string = input("Enter target string: ").strip()

derivation_sequence = generate_derivation(productions, start_symbol, target_string)

if derivation_sequence:
    print("\nDerivation Sequence:")
    for step in derivation_sequence:
        print(step)
else:
    print("\nNo derivation sequence found.")
```

Practical 7:. Design a program for creating machine that accepts the string containing a (Given input{a, b}).

Code:

```
class DFA:
    def __init__(self):
        self.state = "q0"

    def run(self, input_string):
        for char in input_string:
            if char not in {'a', 'b'}:
                return "REJECTED: Invalid character"
            if char == "a":
                self.state = "q1"
        return "ACCEPTED" if self.state == "q1" else "REJECTED"

dfa = DFA()
print(f"Result: {dfa.run(input('Enter a string (a, b only): '))}")
```

Practical 8:. Write python code to design a Turing machine to recognize all strings consisting of an even
number of 1's.

Code:

```
class TuringMachine:
    def __init__(self, tape):
        self.tape = list(tape) + ["_"]
        self.head, self.state = 0, "q0"
        self.transitions = {
            ("q0", "1"): ("q1", "1", "R"),
            ("q1", "1"): ("q0", "1", "R"),
            ("q0", "_"): ("q_accept", "_", "N"),
            ("q1", "_"): ("q_reject", "_", "N")
        }

    def run(self):
        while self.state not in {"q_accept", "q_reject"}:
            symbol = self.tape[self.head]
            if (self.state, symbol) in self.transitions:
                self.state, self.tape[self.head], move = self.transitions[(self.state, symbol)]
                self.head += 1 if move == "R" else -1 if move == "L" else 0
        return "ACCEPTED" if self.state == "q_accept" else "REJECTED"

user_input = input("Enter a string of '1's only: ")
print(TuringMachine(user_input).run() if set(user_input) <= {"1"} else "Invalid input!")
```

Extra DFA codes:

1.Constructed DFA that accepts set of all strings that start with 0.

Code:

```
class DFA:
    def __init__(self):
        self.state = "q0"
        self.accepting_states = {"q1"}

    def transition(self, char):
        if self.state == "q0":
            self.state = "q1" if char == '0' else "q2"
        elif self.state == "q2":
            self.state = "q2"

    def is_accepted(self, binary_string):
        for char in binary_string:
            self.transition(char)
        return self.state in self.accepting_states

user_input = input("Enter a binary string: ")

print("Accepted" if user_input and set(user_input) <= {"0", "1"} and
DFA().is_accepted(user_input) else "Rejected")
```

2. Construct a DFA that accepts set of all strings over 0, 1 of a length 2.

Code:

```
class DFA:
    def __init__(self):
        self.state = "q0"

    def transition(self, char):
        if self.state == "q0":
            self.state = "q1"  # First character read
        elif self.state == "q1":
            self.state = "q2"  # Second character read (Accepting state)
        elif self.state == "q2":
            self.state = "q3"  # Trap state (Length > 2)
        elif self.state == "q3":
            self.state = "q3"  # Stay trapped

    def is_accepted(self, binary_string):
```

```
        for char in binary_string:
            if char not in {"0", "1"}:
                return False  # Invalid character
            self.transition(char)
        return self.state == "q2"

user_input = input("Enter a binary string: ")
dfa = DFA()
print("Accepted" if dfa.is_accepted(user_input) else "Rejected")
```

3. Construct a DFA that accepts set of all strings that ends with a.

Code:

```
class DFA:
    def __init__(self):
        self.state = "q0"

    def transition(self, char):
        self.state = "q1" if char == 'a' else "q0"

    def is_accepted(self, string):
        for char in string:
            if char not in {'a', 'b'}:
                return False  # Invalid input
            self.transition(char)
        return self.state == "q1"  # Accept if in q1

user_input = input("Enter a string (a/b only): ")
dfa = DFA()
print("Accepted" if dfa.is_accepted(user_input) else "Rejected")
```

4.Construct a DFA that accepts set of all strings over A, B that contains a string AABB

Code:

```
class DFA:
    def __init__(self):
        self.state = "q0"

    def transition(self, char):
        if self.state == "q0":
            self.state = "q1" if char == "A" else "q0"
        elif self.state == "q1":
            self.state = "q2" if char == "A" else "q0"
```

```
        elif self.state == "q2":
            self.state = "q3" if char == "B" else "q2"  # AA seen, reset on A
        elif self.state == "q3":
            self.state = "q4" if char == "B" else "q0"  # AAB seen, needs B
        elif self.state == "q4":  # Accepting state
            self.state = "q4"

    def is_accepted(self, input_string):
        for char in input_string:
            if char not in {'A', 'B'}:
                return "REJECTED: Invalid character"
            self.transition(char)
        return "ACCEPTED" if self.state == "q4" else "REJECTED"

dfa = DFA()
user_input = input("Enter a string (A, B only): ")
print(f"Result: {dfa.is_accepted(user_input)}")
```

5. Construct a DFA that not containing a string AABB.

Code:

```
class DFA:
    def __init__(self):
        self.state = "q0"

    def transition(self, char):
        if self.state == "q0":
            self.state = "q1" if char == "A" else "q0"
        elif self.state == "q1":
            self.state = "q2" if char == "A" else "q0"
        elif self.state == "q2":
            self.state = "q3" if char == "B" else "q2"
        elif self.state == "q3":
            self.state = "q4" if char == "B" else "q0"
        elif self.state == "q4":  # Trap state
            self.state = "q4"

    def is_accepted(self, input_string):
        for char in input_string:
            if char not in {'A', 'B'}:
                return "REJECTED: Invalid character"
            self.transition(char)
        return "REJECTED" if self.state == "q4" else "ACCEPTED"
```

```
dfa = DFA()
user_input = input("Enter a string (A, B only): ")
print(f"Result: {dfa.is_accepted(user_input)}")
```