PRACTICAL-1

TOKENIZATION

```
stmt = input('Enter the statement :')
sep=[' ', '(', ')', '[', ']', "'", '"']
b=[]
word=' '
for i in stmt :
    if i not in sep:
        word= word+i
    if i in sep:
        if word != '':
            b.append(word)
        word=' '
        if i!=' ':
            b.append(i)

print(b)
```

OUTPUT:

```
Enter the statement: print('hello')

['print', '(', ' ', '"', 'hello', '"', ' ', ')']
```

GRAMMAR

PRACTICAL-2

```
import re
class StringMethod:
    def __init__(self, my_string):
        self.my_string = my_string
        self.String_arr = []
    def SplitString(Self, sep):
        Self.sep = sep
        temp = []
        for char in self.my_string:
            if char == self.sep:
                val = ''.join(temp)
                self.string_arr.append(val)
                temp.clear()
            else:
                temp.append(char)
        if len(temp) != 0:
            val = ''.join(temp)
            self.string_arr.append(val)
            temp.clear()
        return self.string_arr

if __name__ == '__main__':
    myString = StringMethod('S:AB , A:a , B:b')
    print (myString.SplitString(':'))

def form_dict(arr):
    my_dict = {}
    for prod in arr:
        myString = StringMethod(prod)
        KeyVal = myString.SplitString(':')
        if KeyVal[0] in my-dict.keys():
            if type(my_dict[KeyVal[0]]) != list:
                my-dict[KeyVal[0]] = [my_dict[KeyVal[0]]]
                my_dict[KeyVal[0]].append(KeyVal[1])
            else:
                my_dict[KeyVal[0]].append(KeyVal[1])
        else:
            my_dict[KeyVal[0]] = KeyVal[1]
    return my_dict

def find terminal(productions):
    def check lower(st):
        if st.islower():
            return st
        else:
            for char in st:
                if char.isupper():
                    st = re.sub(char, productions[char], st)
                    return check_lower(st)
    terminals = []
    for var in productions['S']:
        if var.islower():
            terminals.append(var)
        else:
            term = check-lower(var)
            if term not in terminals:
                terminals.append(term)
    return terminals

G = {'nonTerminal': [], 'terminal': [], 'start': [], 'production': []
query = input('Enter Production Rule: ')
```

```
Prods_arr = StringMethod(query)
Prods_arr = Prods_arr.SplitString(',')
Prods = form_dict(Prods_arr)

G('terminals'] = find_ferminal(Prods)
G['nonTerminal'] = list(Prods.key())[1:]
G['start'] = list(Prods.keys())[0]
G['productions'] = Prods

print('\n', G)
```

UTPUT:

```
Enter the Production Rule: S:AB, A:a, B:b

{'nonTerminal':['A','B'], 'terminals':['a','b'], 'start':'S',
 'productions':{'S':'AB', 'A':'a', 'B':'b'}}
```

PRACTICAL-3

```
production = {
        'S' : 'aAC',
        'A' : 'Bb',
        'B' : 'b',
}
key = production.keys()
string = input ('Enter String : ')
isFounded = False
start = 0
subStr = False
tow_is = False

while start < len(string):
    iStr = ' '
    for n in range(start, len(string)).:
        iStr += string[n]
        for k in key :
            if iStr == production[k]:
                string = string.replace(iStr, k, 1)
                start = 0
                subStr = True
                tow_is = True
                break
        if subStr:
            subStr = False
            break
    if tow_is:
        tow_is = False
        continue
        if string == 'S':
            isFounded = True
            break
        start = start +1

if isFounded :
    print ('FOUND', string)
else:
    print ('NOT FOUND', string).
```

OUTPUT:

```
Enter String : abbc
FOUND S

Enter String : abac
NOT FOUND aBac.
```

PRACTICAL-4

TURING MACHINE

```
import re
states = ‘ ’
states = input (‘Enter a string containing states: ’)
statt = inbut (‘Which is your start state ? : ’)
end = input (‘What are your final state(s)? : ’)

terminals = ‘ ’
terminals = input (‘Enter a string containing variables : ’)

tm = { }
for state in states
    tm [state] = { }
    for terminal in terminals:
        tm[state] [terminal] = ‘-’
print(tm)

tm = { ‘A’ : { ‘0’ : ‘B, x, R’ , ‘b’: ‘E, b, R’ };
      ‘B’ : {‘0’ : ‘B, 0, R’, ‘1’, ‘C, y, L’ , ‘y’ : ‘B, y, R’ },
      ‘C’ : {‘0’ : ‘D, 0, L’, ‘x’ : ‘E, x, R’, ‘y’ : ‘C, y, L’ },
      ‘D’ : {‘0’ : ‘D, 0, L’, ‘x’ : ‘A, x, R’ },
      ‘E’ : { ‘y’ : ‘E, y, R’ . ‘b’ ; ‘F, b, R’ },
      ‘F’ : { }
    }

userstring = input (‘enter a string to validate : ’)
userstring = start + userstring
currstate = start
```

!

```
while True :
    print ('Current String -1 ', userstring)
    stringparts = re.split (currstate, userstring)
    if(stringparts[1]== ''):
        stringparts[1]= 'b'
    if (currstate in end) and (stringparts[1]=='b')):
        print ('successfully parsed')
        break
    print ('currstate :', currstate)
    try:
        if (tm [currstate] [stringparts[1][0]]=='-'):
            print ('can't proceed further ...')
            break
    except:
        print ('can't prodeed further ...')
        break
    rule = tm[currstate][ stringparts[1][0]]
    ruleparts = rule.split (',')
    currstate = ruleparts [0]
    stringparts[1] = ruleparts [1] + stringparts [1][1:
    if (ruleparts[2]== 'L'):
        stringparts [0]= stringparts[0] [:len (stringparts[0]
                       + currstate+ stringparts[0][len(
                              stringparts[0]-1 ]
    if(ruleparts[2]== 'R'):
        stringparts[0]= stringparts[1][0]+currstate +
                        stringparts[1][1:]

    userstring= stringparts[0]+ stringparts[1]
```

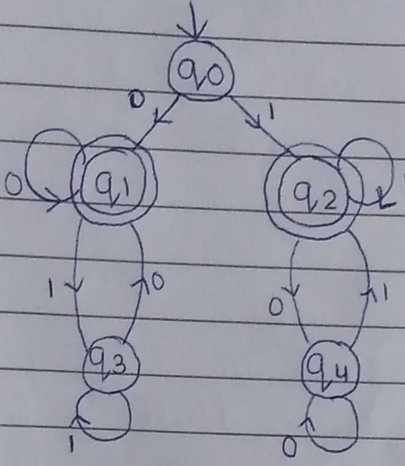
UTPUT

```
Enter a string containing states. ABCDEF
Which is your start state? : A
What are your final state? : F
Enter string containing variables : 01bxy

{'A':{'0':'-', '1':'-', 'b':'-', 'x':'-', 'y':'-'},
  'B':{'0':'-', '1':'-', 'b':'-', 'x':'-', 'y':'-'},
  'C':{'0':'-', '1':'-', 'b':'-', 'x':'-', 'y':'-'},
  'D':{'0':'-', '1':'-', 'b'='-', 'x':'-', 'y':'F'},
  'E':{'0':'-', '1':'-', 'b':'-', 'x';'-', 'y':'-'},
  'F':{'0':'-', '1':'-', 'b':'-', 'x':'-', 'y' '-'}
}

Enter a string to validate : 01b
 Current String -> A01b
 Currstate : A
 Current String -> xB1b
 Curr state : B
 Current String -> Cxyb
 Currstate : C
 Current String -> xEyB
 Currstate : E
 Current~~State~~String -> xyEb
 Currstate : E
 Current String -> xybF
Successfully parsed ...
```

PRACTICAL- 5

DETERMINISTIC FINITE AUTOMATA

Q $L = \{ w \in \{0,1\}^* \mid w$ begin and ends with same symbol$\}$



$Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0 → start state

q_3, q_2 → accept state (ie $F = \{q_1, q_2\}$)

q_1 → begin with 0 and ends with 0

q_2 → begin with 1 and ends with 1

q_3 → begin with 0 and ends with 1

q_4 → begin with 1 and ends with 0.