"Heaven's light is our guide"

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

Lab Report

<u>Course No:</u> CSE 2206

<u>Course Title:</u> Finite Automata Theory Sessional

Submitted By:

SAIFUR RAHMAN

Roll No: 1703018

Section: '17-A

Class:2nd year (Even Semester)

Submitted To:

MD. FARUKUZZAMAN FARUK

Lecturer,

Dept. of CSE,

RUET.

Date of Submission: 9th Nov, 2020

INDEX

LAB 1

Date of Experiment: 31st December, 2019

Date of Submission: 25th January, 2020

PROBLEM NO: 01

Phone Number Checking Problem

PROGRAM TITLE:

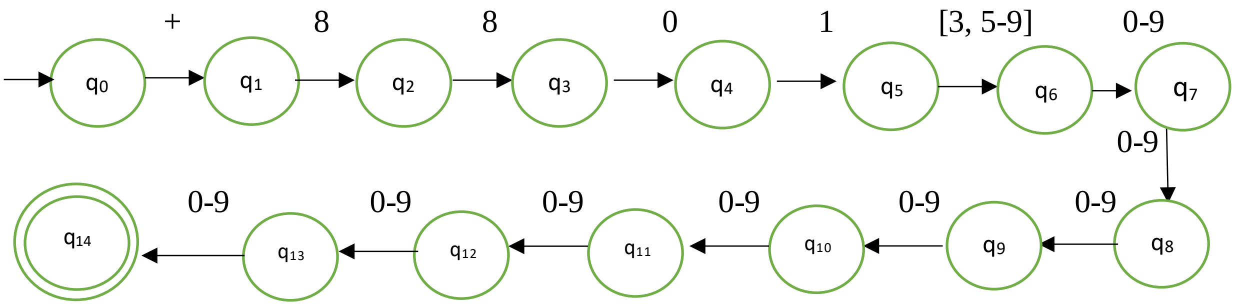
Write a program which checks valid phone numbers from a set of data from file.

OBJECTIVE:

To learn phone number checking system using **Finite Automata**.

THEORY:

The following program can be solved by the corresponding **DFA**:



1. Checks number length is 14 or not.

2. Checks country code.

3. Checks network operator.

4. Checks other digits are correct or not.

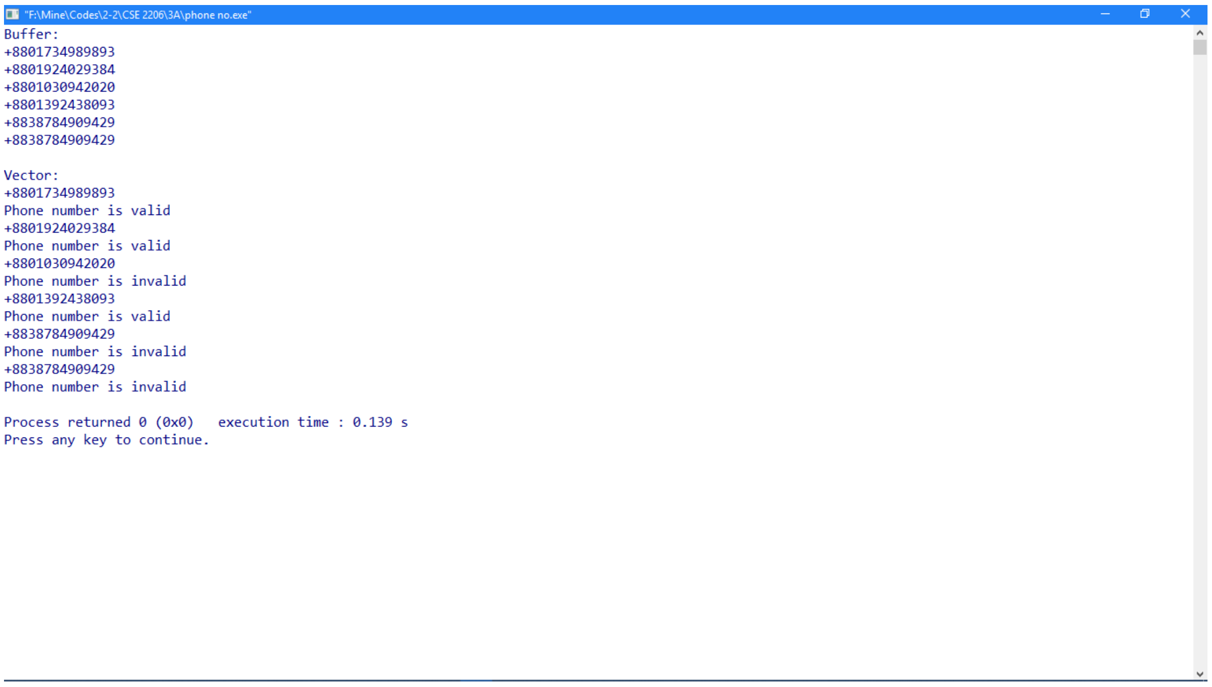
CODE:

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;
string buffer, temp;
vector<string>phoneNo;
int phoneNoAutomata (string phoneNumber);

int main()
```

```
    {
        ifstream f1;
        f1.open("phone no.txt");
        cout << "Buffer:\n";
        while(!f1.eof()){
            f1 >> buffer;
            phoneNo.push_back(buffer);
            cout << buffer << endl;
        }
        cout << "\nVector:\n";
        for(int i = 0; i < phoneNo.size(); i ++){
            cout << phoneNo[i] << endl;
            phoneNoAutomata(phoneNo[i]);
        }
    }
    int phoneNoAutomata (string phoneNumber)
    {
        string phoneNumberFirstPart = phoneNumber.substr(0, 5);
        int flag = 1;
        for(int i = 1; i < phoneNumber.size(); i++){
            if(!isdigit(phoneNumber[i]))
                flag = 0;
        }
        if(phoneNumber.size() != 14)
            flag = 0;
        if(phoneNumberFirstPart != "+8801")
            flag = 0;
        if(phoneNumber[5] == '0' || phoneNumber[5] == '1' || phone
Number[5] == '2' || phoneNumber[5] == '4')
            flag = 0;
        if(flag == 0)
            cout << "Phone number is invalid\n";
        else
            cout << "Phone number is valid\n";
    }
```

OUTPUT:



```
Buffer:
+8801734989893
+8801924029384
+8801030942020
+8801392438093
+8838784909429
+8838784909429

Vector:
+8801734989893
Phone number is valid
+8801924029384
Phone number is valid
+8801030942020
Phone number is invalid
+8801392438093
Phone number is valid
+8838784909429
Phone number is invalid
+8838784909429
Phone number is invalid

Process returned 0 (0x0)   execution time : 0.139 s
Press any key to continue.
```

PROBLEM NO: 2

OTP generation

PROGRAM TITLE:

Write a program to generate some **OTP** of 6 digits like “RUET –XXXXXX”.

OBJECTIVE:

To learn to generate **OTP**.

THEORY:

A one-time password (**OTP**) is an automatically generated numeric or alphanumeric string of characters that authenticates the user for a single transaction or login session. Here **String catenation** is used for “RUET- “substring and **Modular Congruence** method for generating random OTPs.

$$\boldsymbol{x_{n+1} = (ax_n + c)\ mod\ m}$$

$m, 0 < m - the\ "modulus"$

$a, 0 < a < m - the\ "multiplier"$

$c, 0 \leq c < m - the\ "increment"$

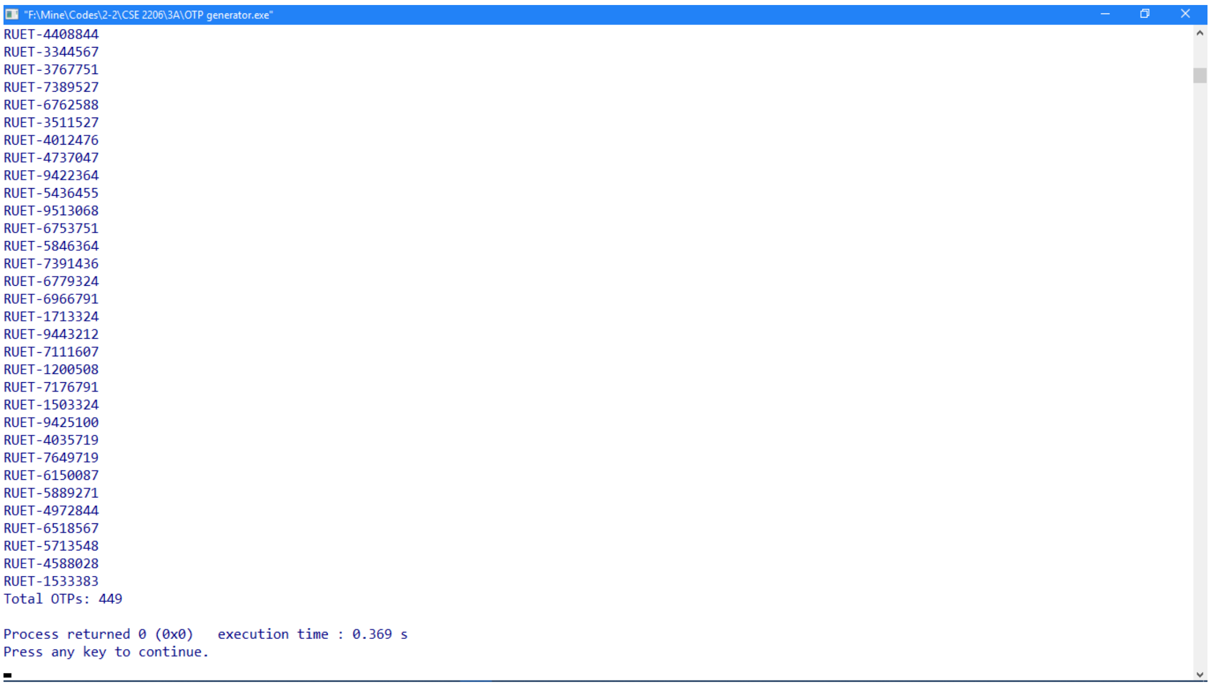
$x_0, 0 \leq x_0 < m - the\ "seed"\ or\ "start\ value"$

CODE:

```cpp
#include <iostream>
#include <fstream>

using namespace std;
int main()
{
    ofstream fout;
    fout.open("OTP.txt");
    int a, c, count = 0;
          long int m, x0, xN;
          a = 1999, c = 3, m = 10000000, x0 = 7;
          for(int i = 0; i < 1000; i++){
              xN = ((a * x0) + c) % m;
              if(xN > 999999 && xN < 9999999){
                  count++;
                  fout << "RUET-" <<  xN << "\n";
                  cout << "RUET-" << xN <<  "\n";
              }
              x0 = xN;
          }
          cout << "Total OTPs: " << count << endl;
      }
```

OUTPUT:



```
RUET-4408844
RUET-3344567
RUET-3767751
RUET-7389527
RUET-6762588
RUET-3511527
RUET-4012476
RUET-4737047
RUET-9422364
RUET-5436455
RUET-9513068
RUET-6753751
RUET-5846364
RUET-7391436
RUET-6779324
RUET-6966791
RUET-1713324
RUET-9443212
RUET-7111607
RUET-1200508
RUET-7176791
RUET-1503324
RUET-9425100
RUET-4035719
RUET-7649719
RUET-6150087
RUET-5889271
RUET-4972844
RUET-6518567
RUET-5713548
RUET-4588028
RUET-1533383
Total OTPs: 449

Process returned 0 (0x0)   execution time : 0.369 s
Press any key to continue.
```

LAB 2

Date of Experiment: 25th January, 2020

Date of Submission: 10th February, 2020

PROBLEM NO: 01

Python Basic Programs

PROGRAM TITLE:

Working with **python basic syntax, programs, loops, conditions, console input output** etc.

OBJECTIVES:

To know about **python basic programs**.

THEORY:

Python:

A simple **language** which is easier to learn. **Python** has a very simple and elegant syntax.

- Free and open-source.
- Portability.
- Extensible and Embeddable.
- A high-level, interpreted **language**.
- Large standard libraries to solve common tasks.
- Object-oriented.

1.1 print():

The **print()** function prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object will be converted into a string before written to the screen.

CODE:

```python
print('Welcome all to python RE')
```

OUTPUT:

```
Welcome all to python RE
```

In this program, we have used the built-in **print()** function to print the string “Welcome all to Python RE” on our screen.

1.2 input():

The **input()** function reads a line entered on a console by an input device such as a keyboard and convert it into a string and returns it. One can use this input string in **python** code.

CODE:

```
a = input('Enter whatever u want: ')
print(a)
print(type(a))
aa = int(a)
print(type(aa))
```

OUTPUT:

```
Enter whatever u want: 50

50

<class 'str'>

<class 'int'>
```

Whatever we enter as input, input function converting it into a string. if we enter an integer value still **input()** function convert it into a string. We need to explicitly convert it into an integer in our code using typecasting.

1.3 for loop:

for loops are traditionally used when you have a block of code which you want to repeat a fixed number of times. The **Python** for statement iterates over the members of a sequence in order, executing the block each time.

CODE:

```
for i in range(0, 10, 1):
    if i % 2 == 0:
        print(i, end = ' ' )
```

OUTPUT:

```
0 2 4 6 8
```

The **range()** function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: **range(0,10,1)**, which means values from 0 to 10 (but not including 10).

PROBLEM NO: 02

NumPy Module in Python

PROGRAM TITLE:

Using and analyzing **NumPy** library in **python**.

OBJECTIVES:

To learn about **NumPy** library in **python**.

THEORY:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

2.1 Array & Dimensions:

A **NumPy** array is a homogeneous block of data organized in a multidimensional finite grid. All elements of the array share the same data type, also called **dtype** (integer, floating-point number, and so on). The **dimensions** of an **array** can be accessed via the "**shape**" attribute that returns a tuple describing the length of each **dimension**.

CODE:

```python
import numpy as np
#numpy
mat1 = np.array( [ (1, 2, 3), (4, 5, 6), (7, 8, 9), (7, 4, 1) ] )
print(mat1)
dimension = mat1.shape
print(dimension[0])
print(dimension[1])
print(dimension)

mat_zeros = np.zeros((4, 3))
mat_ones = np.ones((3, 4))
print(mat_zeros)
print(mat_ones)

print(type(mat_zeros))
print(type(mat_zeros[1][1]))

```

```
for i in range(0, mat_zeros.shape[0]):
    for ii in range(0, mat_zeros.shape[1]):
        mat_zeros[i][ii] = i + ii

print(mat_zeros)
```

OUTPUT:

```
[[1 2 3]

[4 5 6]

[7 8 9]

[7 4 1]]

4

3

(4, 3)

[[0. 0. 0.]

[0. 0. 0.]

[0. 0. 0.]

[0. 0. 0.]]

[[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]]

<class 'numpy.ndarray'>

<class 'numpy.float64'>

[[0. 1. 2.]

[1. 2. 3.]

[2. 3. 4.]

[3. 4. 5.]]
```

2.2 NumPy Matrix Multiplication:

numpy.dot(*a*, *b*, *out=None*)

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If either *a* or *b* is 0-D (scalar), it is equivalent to multiply and using numpy.multiply(a, b) or a * b is preferred.

- If *a* is an N-D array and *b* is a 1-D array, it is a sum product over the last axis of *a* and *b*.
- If *a* is an N-D array and *b* is an M-D array (where M>=2), it is a sum product over the last axis of *a* and the second-to-last axis of *b*:

dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])

CODE:

```python
import numpy as np
#numpy
mat1 = np.array( [ (1, 2, 3), (4, 5, 6), (7, 8, 9), (7, 4, 1) ] )
print(mat1)
dimension1 = mat1.shape
print(dimension1)
mat2 = np.array( [ (1, 2, 3, 4), (4, 5, 7, 8), (9, 6, 3, 1) ] )
print(mat2)
dimension2 = mat2.shape
print(dimension2)
mat3 = np.dot(mat1, mat2)
print(mat3)
print(mat3.shape)
```

OUTPUT:

```
[[1 2 3]
[4 5 6]
[7 8 9]
[7 4 1]]
(4, 3)
[[1 2 3 4]
[4 5 7 8]
[9 6 3 1]]
(3, 4)
[[ 36 30 26 23]
[ 78 69 65 62]
[120 108 104 101]
[ 32 40 52 61]]
(4, 4)
```

PROBLEM NO: 03

re Module in Python

PROGRAM TITLE:

Using and analyzing **RE module** in **python**.

OBJECTIVES:

To learn about **RE module** in **python**.

THEORY:

Regular expressions (RE) use the backslash character ('\') to indicate special forms or to allow special characters to be used without invoking their special meaning. This collides with Python's usage of the same character for the same purpose in string literals; for example, to match a literal backslash, one might have to write '\\\\' as the pattern string, because the regular expression must be \\, and each backslash must be expressed as \\ inside a regular Python string literal. Also, please note that any invalid escape sequences in Python's usage of the backslash in string literals now generate a DeprecationWarning and in the future this will become a SyntaxError. This behaviour will happen even if it is a valid escape sequence for a regular expression.

The solution is to use Python's raw string notation for regular expression patterns; backslashes are not handled in any special way in a string literal prefixed with 'r'. So r"\n" is a two-character string containing '\' and 'n', while "\n" is a one-character string containing a newline. Usually patterns will be expressed in Python code using this raw string notation.

3.1 re.compile():

The **re.compile**(pattern, repl, string): We can combine a regular expression pattern into pattern objects, which can be used for pattern matching. It also helps to search a pattern again without rewriting it.

CODE:

```
import re

data = '''''
Hello everyone, I am MrA.In 11/Jan/2020 I met Messi and got his phone number +8801728366059.But his manager MrAugust
gave me another number +8801987533108 and requested to enjoy a football match at 20/Mar/2021 and I accepted his offer.
'''

date_pattern = re.compile(r'\d+/[A-Z][a-z]+/\d{4}')

scraping_dates = date_pattern.findall(data)
print(scraping_dates)

phone_number_pattern = re.compile(r'\+[8][8][0][1][3|5|6|7|8|9]\d{8}')
scraping_numbers = phone_number_pattern.findall(data)
```

```
print(scraping_numbers)

name_pattern = re.compile(r' [A-Z][a-z][A-Z]*[a-z]*')
scraping_names = name_pattern.findall(data)
print(scraping_names)
```

OUTPUT:

```
['11/Jan/2020', '20/Mar/2021']
['+8801728366059', '+8801987533108']
[' MrA', ' Messi', ' MrAugust']
```

3.2 re.findall():

re.findall() module is used when you want to iterate over the lines of the file, it will return a list of all the matches in a single step.

CODE:

```
#re Personal Assesment
import numpy as np

mat1 = np.array( ['+8801738663624', '+88017205266h20', '+8801720526652', '+8801738663i245', '+8801776604777', '+880177o6047778' ] )
phone_number_pattern = re.compile(r'\+[8][8][0][1][3|5|6|7|8|9]\d{8}')


for i in range(0, mat1.shape[0]):
    if phone_number_pattern.findall(mat1[i]):
        print('Valid number: ' + mat1[i])
    else:
        print('invalid number: ' + mat1[i])
```

OUTPUT:

```
Valid number: +8801738663624

invalid number: +88017205266h20

Valid number: +8801720526652

invalid number: +8801738663i245

Valid number: +8801776604777

invalid number: +880177o6047778
```

LAB 3

Date of Experiment: 10th February 2020

Date of Submission: 3rd March 2020

PROBLEM NO: 01

Factorial determination using Python Functions.

PROGRAM TITLE:

Calculating **factorial** of an integer using **user defined python function**.

OBJECTIVES:

To calculate factorial of an integer.

THEORY:

In all programming and scripting language, a function is a block of program statements which can be used repetitively in a program.

- In Python, a user-defined function's declaration begins with the keyword def and followed by the function name.
- The function may take arguments(s) as input within the opening and closing parentheses, just after the function name followed by a colon.
- After defining the function name and arguments(s) a block of program statement(s) start at the next line and these statement(s) must be indented.

```
def function_name(argument1, argument2, ...) :
      statement_1
      statement_2
      ....
```

CODE:

```python
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

while True:
    n = int(input('Enter a number: '))
    if n == 0:
        break
    else:
        fact = factorial(n)
        print('Factorial of ', n, ' is ', fact)
```

OUTPUT:

```
Enter a number: 10
Factorial of  10  is  3628800
Enter a number: 0
```

PROBLEM NO: 02

String Matching

PROGRAM TITLE:

Write a python program that matches a string that has an 'a' followed by anything ending 'b'.

OBJECTIVES:

To learn string matching using regex.

THEORY:

2.1 re.search ():

The search() function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned.

CODE:

```python
import re
import numpy as np

def patern_match(text):
    patterns = 'a.*b$'
    if re.search(patterns, text):
        return 1
    else:
        return 0

data = np.array(['ab', 'aabb', 'abcd', 'abababc', 'darb', 'bab', 'sa job'])

for i in data:
    flag = patern_match(i)
    if flag == 1:
        print(i, ' is matched')
    else:
        print(i, 'is not matched')
```

OUTPUT:

```
ab  is matched
aabb  is matched
abcd is not matched
abababc is not matched
darb  is matched
bab  is matched
sa job  is matched
```

PROGRAM TITLE:

Write a python program that finds scores and wickets of two teams from a given text.

THEORY:

2.2 re.split ():

The split() function returns a list where the string has been split at each match

CODE:

```python
data = ''''India played U-19 world cup final in 2006, 2008, 2010, 2020. Top scorers from BD U-19 team was Akbar Ali 43(59), Parvez 47(79), Rokibul 20(32). And from bowlers end Sakib 3-28, Shariful 2-51, Rokibul 2-37. And top players from IND U-19 team was Joysowal 88(121), Saxena 2(17), Veer 0(1). and from bowling end Tyagi 0-33, Mishra 2-25. they performed well too.'''

import re

Year_pattern = re.compile(r'\d{4}')
years = Year_pattern.findall(data)
print("IND played world cup finals:")
print(years)

for item in re.findall("\S.*BD.*\w\S", data):
    year, stat = re.split("B", item)

for line in re.findall("\S.*IND.*\w\S", stat):
    ban, ind = re.split("I", line)

Run_pattern = re.compile(r'[0-9]+\([0-9]+\)')
Wicket_pattern = re.compile(r'[0-9]\-[0-9]+')

print("\nBD stats:")
BD_runs = Run_pattern.findall(ban)
print(BD_runs)
BD_wickets = Wicket_pattern.findall(ban)
print(BD_wickets)

print("\nIND stats:")
IND_runs = Run_pattern.findall(ind)
```

```python
print(IND_runs)
IND_wickets = Wicket_pattern.findall(ind)
print(IND_wickets)
```

OUTPUT:

```
IND played world cup finals:
['2006', '2008', '2010', '2020']

BD stats:
['43(59)', '47(79)', '20(32)']
['3-28', '2-51', '2-37']

IND stats:
['88(121)', '2(17)', '0(1)']
['0-33', '2-25']
```

2.3 re.sub ():

re.sub() function is used to replace occurrences of a particular **sub**-string with another **sub**-string. This function takes as input the following: The **sub**-string to replace. The **sub**-string to replace with.

CODE:

```python
#re.sub()

sender = re.search("From:.* ", fh)
address = sender.group()

email = re.sub('From', 'Email', address)

print(address)
print(email)
```

OUTPUT:

```
From: "Saifur Rahman"
Email: "Saifur Rahman"
```

LAB 4

Date of Experiment: 26th February 2020

Date of Submission: 3rd March 2020

PROBLEM NO: 01

Implementing DFA

PROGRAM TITLE:

Implementing and analyzing DFA.

OBJECTIVES:

To learn to implement DFA using python.

THEORY:

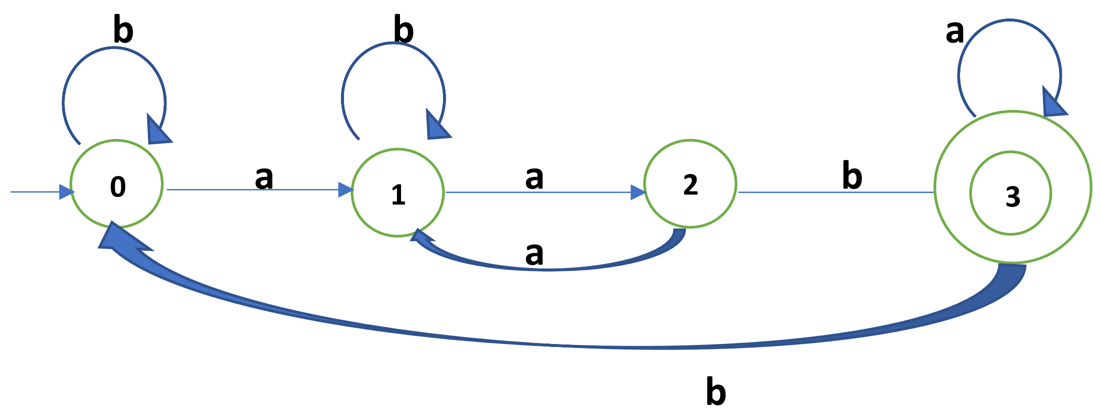
DFA stands for **Deterministic Finite Automaton**. In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**. As it has a **finite number of states**, the machine is called **Deterministic Finite Machine** or **Deterministic Finite Automaton.**

Formal Definition of a DFA

A DFA can be represented by a 5-tuple (Q, ∑, δ, q_0, F) where −

- **Q** is a finite set of states.
- **∑** is a finite set of symbols called the alphabet.
- **δ** is the transition function where δ: Q × ∑ → Q
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Example DFA



DFA 5-tuple

Q = {0, 1, 2, 3}

∑ = {a, b}

q_0 = {0}

F = {3}

Transition Function δ showed by following table:

Present State	Next State for Input a	Next State for Input b
→0	1	0
1	2	1
2	1	*3
*3	*3	0

CODE:

```python
#pip install pyformlang

#importing necesssary modules
from pyformlang.finite_automaton import DeterministicFiniteAutomaton
from pyformlang.finite_automaton import State
from pyformlang.finite_automaton import Symbol

#declaration of dfa

dfa = DeterministicFiniteAutomaton()

#Creation as states
state0 = State(0)
state1 = State(1)
state2 = State(2)
state3 = State(3)

#Creation of symbols

sym_a = Symbol("a")
sym_b = Symbol("b")

#Add start state
dfa.add_start_state(state0)

#Add final state
dfa.add_final_state(state3)

#Creation of transition function...
dfa.add_transition(state0, sym_a, state1)
dfa.add_transition(state0, sym_b, state0)
dfa.add_transition(state1, sym_a, state2)
dfa.add_transition(state1, sym_b, state1)
dfa.add_transition(state2, sym_a, state1)
dfa.add_transition(state2, sym_b, state3)
dfa.add_transition(state3, sym_a, state3)
dfa.add_transition(state3, sym_b, state0)

#check if a word is accepted
dfa_result1 = dfa.accepts([sym_a, sym_a, sym_b])
dfa_result2 = dfa.accepts(['a', sym_b, sym_a, sym_b])
dfa_result3 = dfa.accepts(["a", "b", "a", "a", "b"])
dfa_result4 = dfa.accepts(['b', "a", "a", "a"])

print('Results: ', dfa_result1, ' ', dfa_result2, ' ', dfa_result3, ' ', df
a_result4)
```

OUTPUT:

```
Results:   True    True    False    False
```

PROGRAM TITLE:

Implementing and analyzing DFA with list.

THEORY:

List is one of the python data types. **List** is a collection which is ordered and changeable. Allows duplicate members. In Python lists are written with square brackets.

```python
thislist=["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

CODE:

```python
import numpy as np
from pyformlang.finite_automaton import DeterministicFiniteAutomaton
from pyformlang.finite_automaton import State
from pyformlang.finite_automaton import Symbol

#declaration of dfa

dfa = DeterministicFiniteAutomaton()

#Creation as states
state0 = State(0)
state1 = State(1)
state2 = State(2)
state3 = State(3)

#Creation of symbols
```

```

sym_a = Symbol("a")
sym_b = Symbol("b")

#Add start state
dfa.add_start_state(state0)

#Add final state
dfa.add_final_state(state3)

#Creation of transition function...
dfa.add_transition(state0, sym_a, state1)
dfa.add_transition(state0, sym_b, state0)
dfa.add_transition(state1, sym_a, state2)
dfa.add_transition(state1, sym_b, state1)
dfa.add_transition(state2, sym_a, state1)
dfa.add_transition(state2, sym_b, state3)
dfa.add_transition(state3, sym_a, state3)
dfa.add_transition(state3, sym_b, state0)


data = np.array(['aab', 'abab', 'abaab', 'baaa', 'ababab', 'aabbbb', 'aaaab
'])

for i in data:
    list = []
    for j in i:
        list.append(j)

    if dfa.accepts(i):
        print(list, ' is accepted')
    else:
        print(list, 'is not accepted')
```

OUTPUT:

```
['a', 'a', 'b']  is accepted
['a', 'b', 'a', 'b']  is accepted
['a', 'b', 'a', 'a', 'b'] is not accepted
['b', 'a', 'a', 'a'] is not accepted
['a', 'b', 'a', 'b', 'a', 'b'] is not accepted
['a', 'a', 'b', 'b', 'b', 'b'] is not accepted
['a', 'a', 'a', 'a', 'b']  is accepted
```