**Answer Key**

**B2-SET A**

**PART B**

**11.**

From pyDatalog import pyDatalog

pyDatalog.create\_terms('factorial, N')

factorial[N] = N\*factorial[N-1]

factorial[1] = 1

print(factorial[3]==N)

**12.**

from pyDatalog import pyDatalog

pyDatalog.create\_terms('X')

print(X.in\_((0,1,2,3,4)))

**13.**

**Logic Programming Paradigm:**

Logic programming is a paradigm where computation arises from proof search in a logic according to a fixed, predictable strategy. A logic is a language. It has syntax and semantics. It. More than a language, it has inference rules.

A Logic program is a set of predicates. Ex parent, siblings

pyDatalog is the library that we use to create logic in python. The example of such a program is:

from pyDatalog import pyDatalog

pyDatalog.create\_terms(‘X,Y,Z’)

+X(‘A’, ‘10’)

**Dependent Programming Paradigm:**

Writing a correct computer program is hard and proving that a program is correct is even harder. Dependent Types allow us to write programs and know they are correct before running them. you can specify types that can check the value of your variables at compile time.

A function has dependent type if the type of a function's result depends on the VALUE of its argument; this is not the same thing as

a Parameterized Type.

There are two different types of logics used in Dependent Programming:

1. Propositional Logic(Facts)
2. Predicate Logic

**14.**

**Datagram Socket:**

Connectionless channel.

Not dedicated & end-to-end channel between server and client.

Use UDP for data transmission.

Not 100% reliable and may lose data.

Data sent/received order might not be the same.

Don't care or rapid recovering lost/mistaken data.

**Stream Socket:**

Connection-Oriented communication.

Dedicated & end-to-end channel between server and client.

Use TCP protocol for data transmission.

Reliable and Lossless.

Data sent/received in the similar order.

Long time for recovering lost/mistaken data

**15.**

from automata.fa.dfa import DFA

dfa = DFA(

*states*={'q0', 'q1'}, *input\_symbols*={'0', '1'},

*transitions*={

'q0': {'0': 'q0', '1': 'q1'},

'q1': {'0': 'q1', '1': 'q0'},

},

*initial\_state*='q0', *final\_states*={'q0'}

)

num = input("Enter the string :")

if(dfa.accepts\_input(num)):

print("Accepted")

else:

print("Rejected")

**PART C**

**16.**

**Server Side:**

import socket

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM, 0)

host = socket.gethostname()

port = 12345

s.bind((host, port))

print(host)

print("Waiting for the connection...")

s.listen(5)

while True:

conn, address = s.accept()

print("Got connection from", address)

data = conn.recv(1024).decode()

data = data[::-1]

conn.send(data.encode())

conn.close()

**Client Side:**

import socket

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM, 0)

host = socket.gethostname()

port = 12345

s.connect((host, port))

print("Connected to server")

while True:

msg = input("Enter your message: ")

s.send(msg.encode())

print("Server: ", s.recv(1024))

s.close()

**Sample Input and Output:**



**17.**

from pyDatalog import pyDatalog

pyDatalog.create\_terms('X, Y, marks, passm, grades, PassM')

+marks('Ram', '96')

+marks('Raju', '49')

+marks('Priya', '86')

+marks('Carol', '78')

+marks('Shyam', '79')

+marks('Maya', '44')

+grades('Ram', 'O')

+grades('Raju', 'F')

+grades('Priya', 'A')

+grades('Carol', 'B')

+grades('Shyam', 'B')

+grades('Maya', 'F')

*# Operation 01*

print(marks(X, Y), "\n")

*# Operation 02*

print(marks(X, '86'), "\n")

*# Operation 03*

print(marks('Priya', Y), "\n")

*# Operation 04*

*# Passm rule that store the students list who failed.*

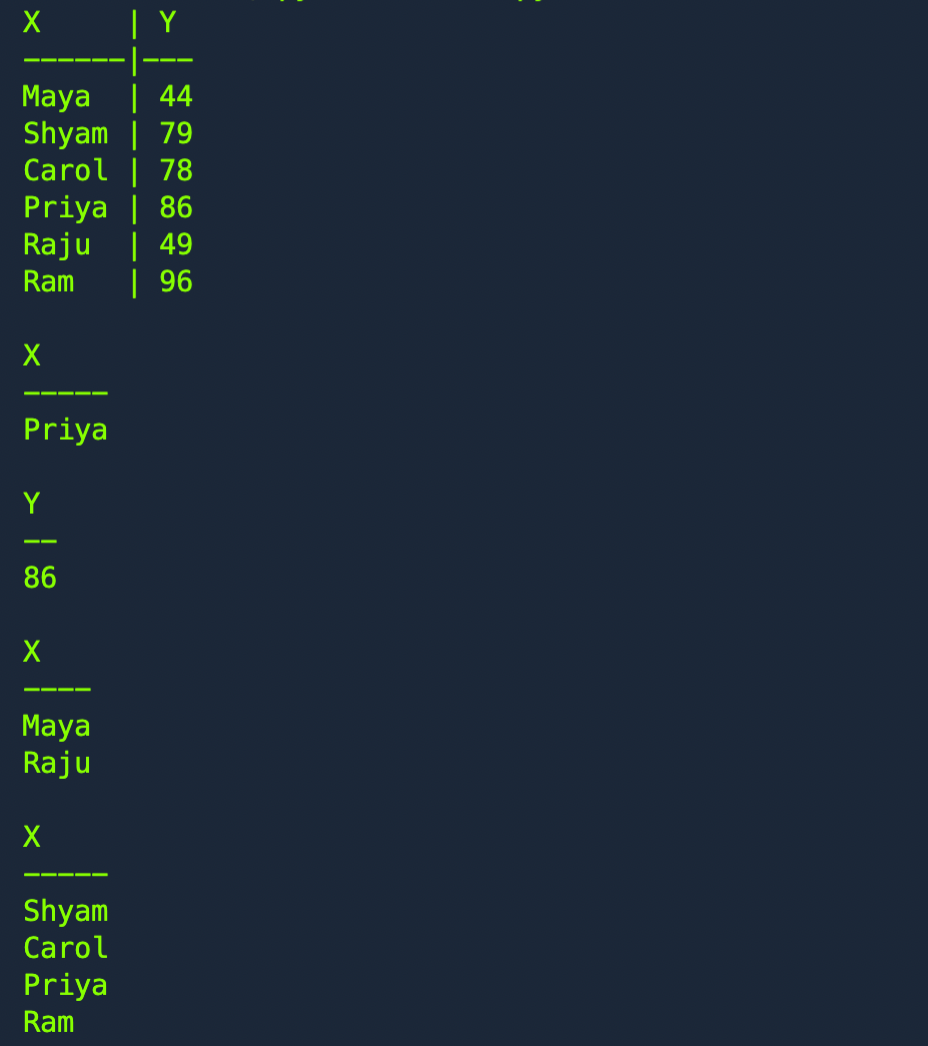
passm(X) <= grades(X, 'F')

print(passm(X), "\n")

*# Operation 05*

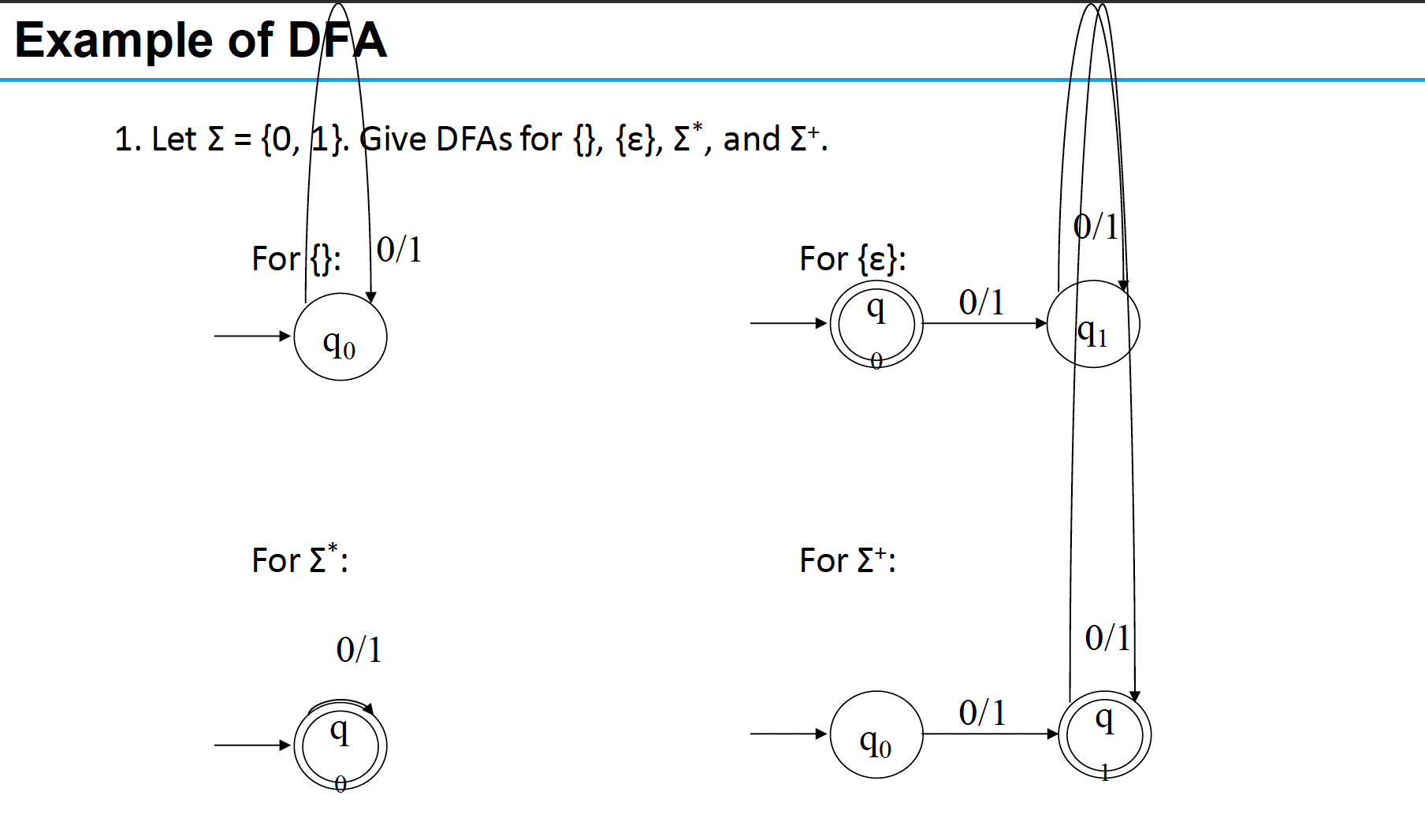
PassM(X) <= marks(X, Y) & ~passm(X)

print(PassM(X), "\n")

**Output: **

**18.**

**a.)**

****

**b.)**

from automata.fa.dfa import DFA

dfa = DFA(

*states*={'q0', 'q1', 'q2', 'q3', 'q4'}, *input\_symbols*={'a', 'b'},

*transitions*={

'q0': {'a': 'q1', 'b': 'q4'},

'q1': {'a': 'q2', 'b': 'q3'},

'q2': {'a': 'q2', 'b': 'q3'},

'q3': {'a': 'q4', 'b': 'q4'},

'q4': {'a': 'q4', 'b': 'q4'}

},

*initial\_state*='q0', *final\_states*={'q1', 'q3'}

)

num = input("Enter the string :")

if(dfa.accepts\_input(num)):

print("Accepted")

else:

print("Rejected")

**Output:**

****

****

**19.**

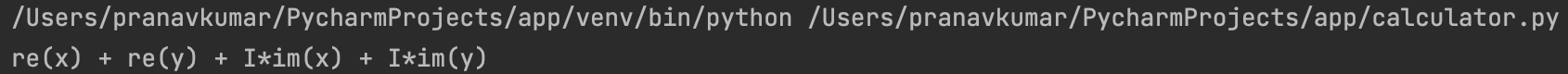
**i.**

from sympy import \*  
x = Symbol('x')  
y = Symbol('y')  
expr = (x + y + x - y)  
print(simplify(expr))

**ii. a**

from sympy import \*  
x = Symbol('x')  
y = Symbol('y')  
print(simplify(I\*im(x) + I\*im(y) + re(x) + re(y)))

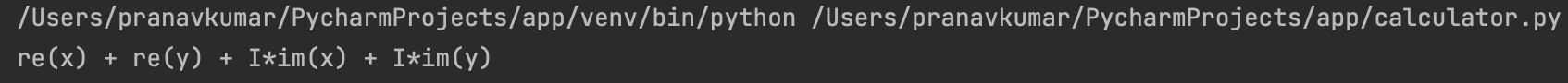
Output:



**ii. b**

from sympy import \*  
x = Symbol('x')  
y = Symbol('y')  
print(expand(x + y, complex=True))

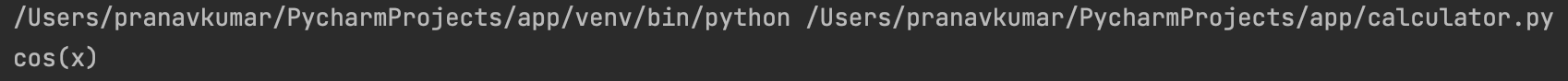
Output:



**ii. c.**

from sympy import \*  
x = Symbol('x')  
# y = Symbol('y')  
print(diff(sin(x), x))

Output:



**iii.**

from sympy import \*  
x = Symbol('x')  
y = Symbol('y')  
A = Matrix([[1, x], [y, 1]])  
print(A)

Output:

