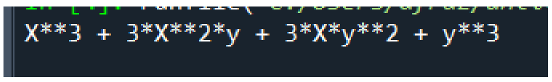
APP CT-3 EXAM B1 – SET B

11)

a)

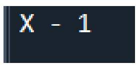
```
import sympy as sym
x=sym.symbols('X')
y=sym.symbols('y')
print(sym.expand((x+y)**3))
```



```
X**3 + 3*X**2*y + 3*X*y**2 + y**3
```

b)

```
import sympy as sym
x=sym.symbols('X')
y=sym.symbols('y')
print(sym.simplify((x**3+x**2-x-1)/(x**2+2*x+1)))
```

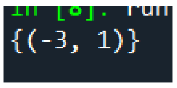


```
X - 1
```

12)

a)

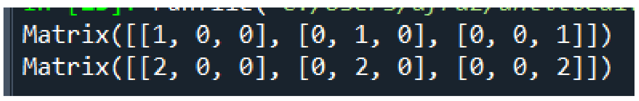
```
import sympy as sym
x= sym.symbols('x')
y= sym.symbols('y')
print(sym.linsolve([x + 5*y - 2,-3*x+6*y -15], (x, y)))
```



```
{(-3, 1)}
```

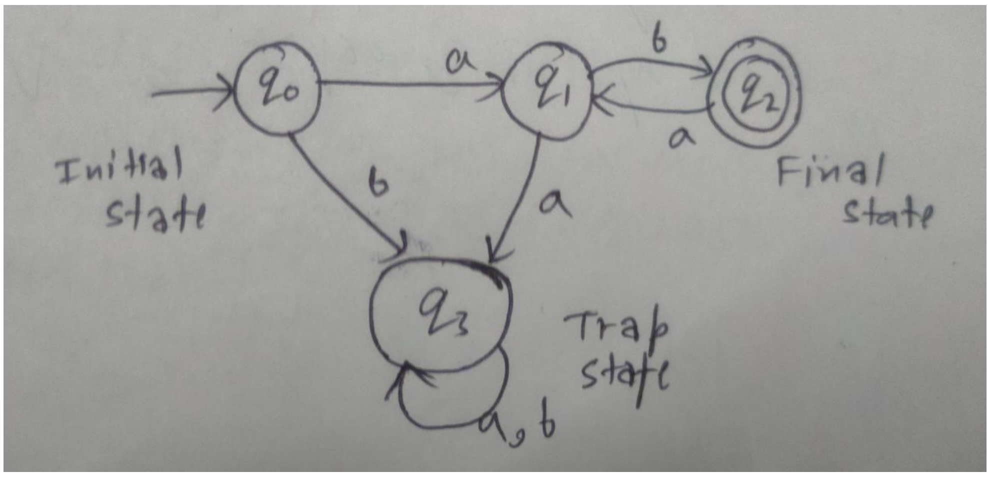
b)

```
import sympy as sym
a=sym.Matrix([[1,0,0],[0,1,0],[0,0,1]])
b=sym.Matrix([[1,0,0],[0,1,0],[0,0,1]])
print(a*b)
print(a+b)
```



```
Matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
Matrix([[2, 0, 0], [0, 2, 0], [0, 0, 2]])
```

13)



14)

a)
bind( ): This method binds the socket to an address. The format of address depends on socket family mentioned above(AF_INET).

b)

listen(backlog) : This method listens for the connection made to the socket. The backlog is the maximum number of queued connections that must be listened before rejecting the connection.

15)
Dependent type 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.
- dependent types: you can specify types that can check the value of your variables at compile time

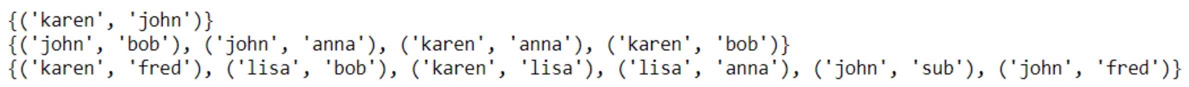
A function has dependent type if the type of a function's result depends on the VALUE of its argument; this is not the same thing as a ParameterizedType. The second order lambda calculus possesses functions with dependent types.

Depends on the application domain, but could mean one or more of:

- Functionally correct (e.g. arithmetic operations on a CPU)
- Resource safe (e.g. runs within memory bounds, no memory leaks, no accessing unallocated memory, no deadlock. . .)
- Secure (e.g. not allowing access to another user’s data)

16)

```
from pyDatalog import pyDatalog
pyDatalog.create_terms('anna,bob,lisa,fred,sub,karen,john,brother,cousin,grandson,descende
nt,X,Y')
+brother('karen','john')
+grandson('karen','anna')
+grandson('karen','bob')
+grandson('john','anna')
+grandson('john','bob')
+descendent('lisa','anna')
+descendent('lisa','bob')
+descendent('karen','lisa')
+descendent('karen','fred')
+descendent('john','fred')
+descendent('john','sub')
print(pyDatalog.ask('brother(X,Y)'))
print(pyDatalog.ask('grandson(X,Y)'))
print(pyDatalog.ask('descendent(X,Y)'))
```

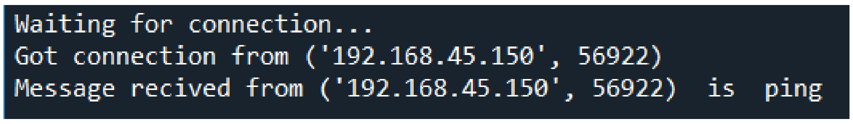


```
{('karen', 'john')}
{('john', 'bob'), ('john', 'anna'), ('karen', 'anna'), ('karen', 'bob')}
{('karen', 'fred'), ('lisa', 'bob'), ('karen', 'lisa'), ('lisa', 'anna'), ('john', 'sub'), ('john', 'fred')}
```

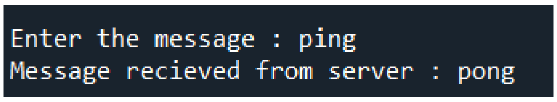
17)

```
#server
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host=socket.gethostname()
port=9999
s.bind((host,port))
print ("Waiting for connection...")
s.listen(5)
while True:
    conn,addr=s.accept()
    print ('Got connection from',addr)
    x=conn.recv(1024).decode("utf-8")
    print("Message recived from",addr," is ",x)
    if(x=="ping"):
        conn.send(bytes('pong',"utf-8"))
    conn.close()
```

```
#client
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host=socket.gethostname()
port=9999
s.connect((host,port))
x=input("Enter the message : ")
s.send(bytes(x,"utf-8"))
y=s.recv(1024).decode("utf-8")
if y:
    print("Message recieved from server :",y)
s.close()
```



```
Waiting for connection...
Got connection from ('192.168.45.150', 56922)
Message recived from ('192.168.45.150', 56922)  is  ping
```

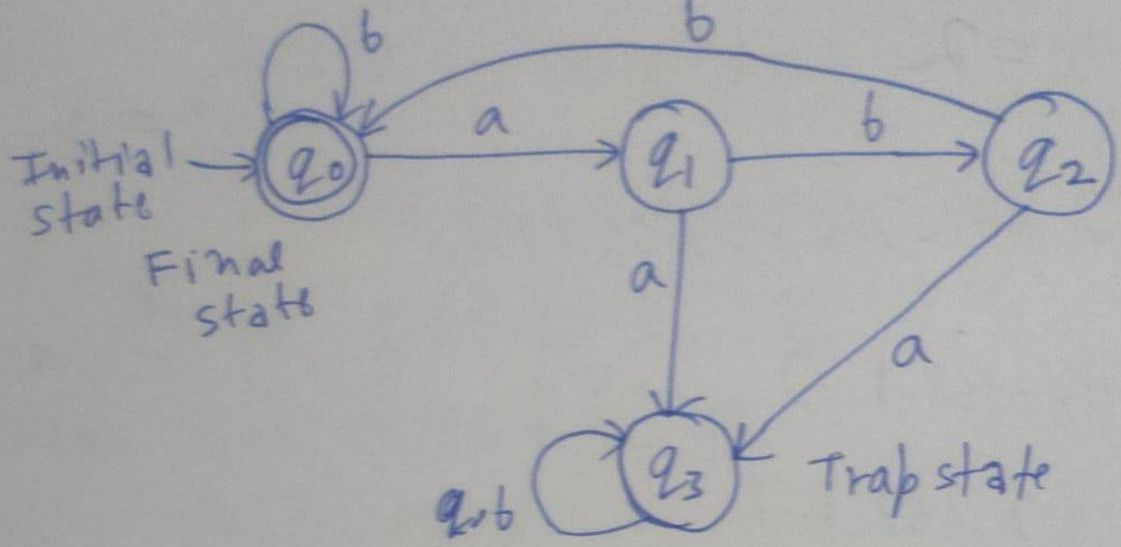


```
Enter the message : ping
Message recieved from server : pong
```

18)

The desired language will be like

$L1 = \{ \varepsilon, abb, abbabb, \ldots \}$



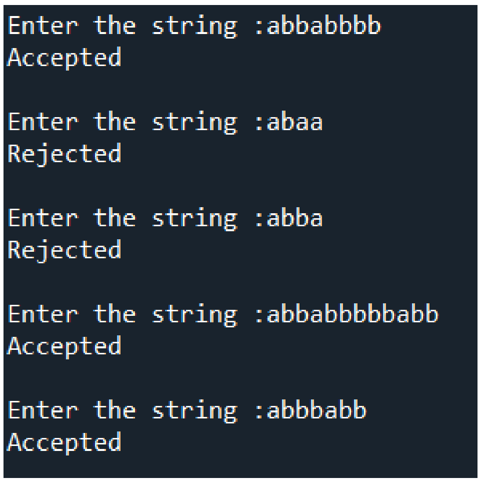
$$\begin{tikzcd}
q_0 \arrow[loop above, "b"] \arrow[r, "a"] & q_1 \arrow[r, "b"] \arrow[d, "a"] & q_2 \arrow[ll, bend right, "b"'] \arrow[dl, "a"] \\
 & q_3 \arrow[loop left, "{a,b}"] &
\end{tikzcd}$$

<u>Transition table:</u>

	a	b
q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3	q_3	q_3

Python code:

```
from automata.fa.dfa import DFA
dfa = DFA(
states={'q0', 'q1', 'q2','q3'}, input_symbols={'a', 'b'},
transitions={
'q0': {'a': 'q1', 'b': 'q0'},
'q1': {'a': 'q3', 'b': 'q2'},
'q2': {'a': 'q3', 'b': 'q0'},
'q3': {'a': 'q3', 'b': 'q3'}
},
initial_state='q0', final_states={'q0'}
)
for i in range(1,6):
    num = input("Enter the string :")
    if(dfa.accepts_input(num)):
        print("Accepted")
    else:
        print("Rejected")
```

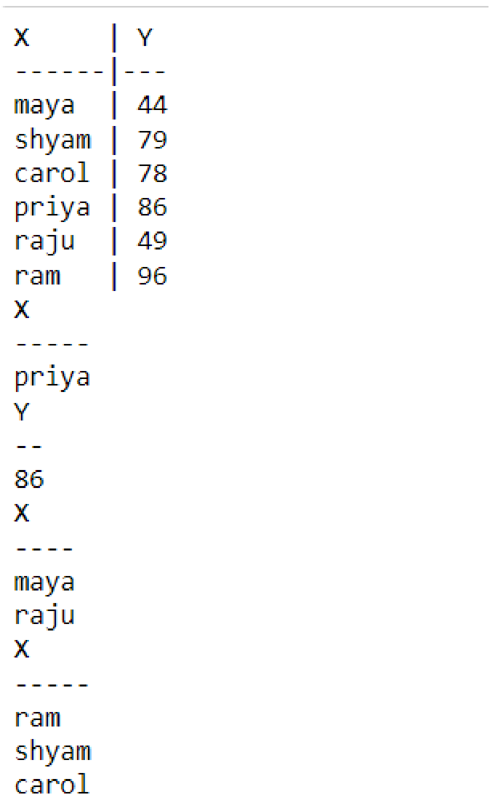


```
Enter the string :abbabbbb
Accepted

Enter the string :abaa
Rejected

Enter the string :abba
Rejected

Enter the string :abbabbbbbabb
Accepted

Enter the string :abbbabb
Accepted
```

19)

```
from pyDatalog import pyDatalog
pyDatalog.create_terms('ram,raju,priya,carol,shyam,maya,X,Y,Z,marks,passm,failm,grades')
+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('shyam','B')
+grades('carol','B')
+grades('maya','F')
print(marks(X,Y))
print(marks(X,'86'))
print(marks('priya',Y))
failm(X)<=grades(X,'F')
passm(X)<=(grades(X,'O') or grades(X,'A') or grades(X,'B'))
print(failm(X))
print(passm(X))
```



```
X      | Y
------|---
maya   | 44
shyam  | 79
carol  | 78
priya  | 86
raju   | 49
ram    | 96
X
-----
priya
Y
--
86
X
----
maya
raju
X
-----
ram
shyam
carol
```