APP week 11 Lab

Q1.

Implement using pyDatalog:
Assume given a set of facts of the form `father(name1,name2)` (`name1` is the father of `name2`), .

a. Define a predicate `brother(X,Y)` which holds iff `X` and `Y` are brothers.
b. Define a predicate `cousin(X,Y)` which holds iff `X` and `Y` are cousins.
c. Define a predicate `grandson(X,Y)` which holds iff `X` is a grandson of `Y`.
d. Define a predicate `descendent(X,Y)` which holds iff `X` is a descendent of `Y`.
e. Consider the following genealogical tree:

What are the answers generated by your definitions for the queries:

```
brother(X,Y)
cousin(X,Y)
grandson(X,Y)
descendent(X,Y)
```

Code:

```
from pyDatalog import pyDatalog
pyDatalog.create_terms('a,b,c,d,e,f,brother, cousin, grandson, descendent, X,Y')
+brother('b', 'c')
+brother('d', 'e')
+cousin('d','f')
+cousin('e','f')
+grandson('d', 'a')
+grandson('e', 'a')
+grandson('f', 'a')
+descendent ('b', 'a')
+descendent ('c','a')
+descendent ('d', 'b')
+descendent('f', 'c')
print(pyDatalog.ask('brother (X,Y)'))
print(pyDatalog.ask('cousin (X, Y)'))
print(pyDatalog.ask('grandson(X, Y)'))
print(pyDatalog.ask('descendent (X, Y)'))
```

Output:

```
{('d', 'e'), ('b', 'c')}
{('d', 'f'), ('e', 'f')}
{('f', 'a'), ('d', 'a'), ('e', 'a')}
{('c', 'a'), ('b', 'a'), ('d', 'b'), ('f', 'c')}
>>>
```

Q2. Encode the following facts and rules in pyDatalog:

- **Bear is big**
- **Elephant is big**
- **Cat is small**
- **Bear is brown**
- **Cat is black**
- **Elephant is gray**
- **An animal is dark if it is black**
- **An animal is dark if it is brown**

Write a query to find which animal is dark and big.

Code:

```
from pyDatalog import pyDatalog
pyDatalog.create_terms('X,Y,Z,bear,elephant,cat,small,big,brown,black,gray,dark')
+big('elephant')
+big('bear')
+small('cat')
+black('cat')
+brown('bear')
+gray('elephant')
dark(X)<=black(X) or brown(X)
print(big(X),dark(X))
```

Output:

```
X
--------
bear
elephant X
---
cat
>>>
```

Q3.

The following are the marks scored by 5 students.

Student Name	Mark
Ram	90
Raju	45
Priya	85
Carol	70
Shyam	80

Enter the above data using pyDatalog.

Write queries for the following:

a. Print Student name and mark of all students.
b. Who has scored 80 marks?
c. What mark has been scored by Priya?
d. Write a rule 'passm' denoting that pass mark is greater than 50. Use the rule to print all students who failed.
e. Write rules for finding grade letters for a marks and use the rule to find the grade letter of a given mark.

Code:

```
from pyDatalog import pyDatalog
pyDatalog.create_terms('X,Y,Z,student,marks,passm,grades')
+student('ram')
+student('raju')
+student('priya')
+student('carol')
+student('shyam')
+marks('90','ram')
+marks('45','raju')
+marks('85','priya')
+marks('70','carol')
+marks('80','shyam')
+grades('ram','O')
+grades('priya','A')
+grades('shyam','A')
+grades('carol','B')
+grades('raju','E')
print(marks(X,Y))
print(marks('80',X))
print(marks(X,'priya'))
passm(X)<=grades(X,'E')
print(passm(X))
```

Output:

```
------------------
X   | Y
---|------
80 | shyam
70 | carol
85 | priya
45 | raju
90 | ram
X
-----
shyam
X
--
85
X
----
raju
>>>
```

Q4. Solve the set of queries in the previous question using imperative programming paradigm in Python. Store the data in a dictionary.

Code:

```
marks={90:'ram',85:'priya',80:'shyam',70:'carol',45:'raju'}
for i in marks:
    print(i,marks[i])
print(marks[80])
for i in marks:
    if marks[i]=='priya':
        print(i)
for i in marks:
    if i<50:
        print(marks[i])
for i in marks:
    if i>=90:
        print(marks[i],'O')
    elif i<90 and i>=80:
        print(marks[i],'A')
    elif i<80 and i>=70:
        print(marks[i],'B')
    elif i<70 and i>=60:
        print(marks[i],'C')
    elif i<60 and i>=50:
        print(marks[i],'D')
    else:
        print(marks[i],'E')
```

Output:

```
90 ram
85 priya
80 shyam
70 carol
45 raju
shyam
85
raju
ram O
priya A
shyam A
carol B
raju E
```

Q5. Write a recursive program to find factorial of a number using pyDatalog.

Code:

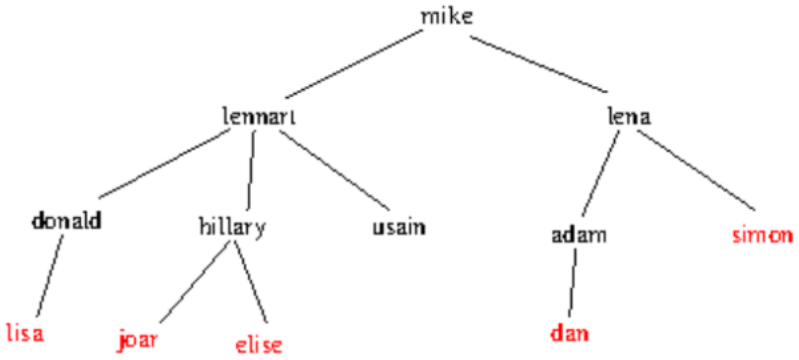
```
from pyDatalog import pyDatalog
pyDatalog.create_terms('factorial, N')
num=int(input('Enter any number:'))
factorial[N] = N*factorial[N-1]
factorial[1] = 1
print(factorial[num]==N)
```

Output:

```
----------------------
Enter any number:3
N
-
6
>>>
```

Q6.

Implement the following using pyDatalog for the family tree where hillary lena, lisa, and elise are female



Define new predicates (in terms of rules using male/1, female/1 and parent/2) for the following family relations:

(a) father	(b) sister	(c) grandmother	(d) cousin	(e) grandfather
(f) mother	(g) brother	(h) uncle	(i) aunty	

Write a query to return the cousin of adam

Write a query to return the grandfather of elise

Code:

```
from pyDatalog import pyDatalog

pyDatalog.clear()

# Define female/1 predicate

pyDatalog.create_terms('female')

female('Hillary')

female('Lena')

female('Lisa')

female('Elise')

# Define male/1 predicate

pyDatalog.create_terms('male')

male('Mike')

male('Lennari')

male('Donald')

male('Usaim')

male('Adam')
```

```
male('Simon')
male('Joar')
male('Dan')
# Define parent/2 predicate
pyDatalog.create_terms('parent')
parent('Mike', 'Lennari')
parent('Mike', 'Lena')
parent('Lennari', 'Donald')
parent('Lennari', 'Hillary')
parent('Lennari', 'Usaim')
parent('Lennari', 'Adam')
parent('Lennari', 'Simon')
parent('Donald', 'Lisa')
parent('Donald', 'Joar')
parent('Hillary', 'Elise')
parent('Usaim', 'Dan')
# Define father/1 predicate
pyDatalog.create_terms('father')
X = pyDatalog.Variable()  # Create X variable
Y = pyDatalog.Variable()  # Create Y variable
father(X, Y) <= male(X) & parent(X, Y)
# Define sister/2 predicate
pyDatalog.create_terms('sister')
X = pyDatalog.Variable()  # Create X variable
Y = pyDatalog.Variable()  # Create Y variable
Z = pyDatalog.Variable()  # Create Z variable
sister(X, Y) <= female(X) & parent(Z, X) & parent(Z, Y) & (X != Y)
# Define grandmother/2 predicate
pyDatalog.create_terms('grandmother')
grandmother(X, Z) <= female(X) & parent(X, Y) & parent(Y, Z)
```

```
# Define cousin/2 predicate
pyDatalog.create_terms('cousin')
cousin(X, Y) <= parent(Z, X) & parent(W, Y) & (Z != W) & sibling(Z, W)
# Define grandfather/2 predicate
pyDatalog.create_terms('grandfather')
grandfather(X, Y) <= male(X) & parent(X, Z) & parent(Z, Y)
# Define mother/1 predicate
pyDatalog.create_terms('mother')
mother(X, Y) <= female(X) & parent(X, Y)
# Define brother/2 predicate
pyDatalog.create_terms('brother')
brother(X, Y) <= male(X) & parent(Z, X) & parent(Z, Y) & (X != Y)
# Define uncle/2 predicate
pyDatalog.create_terms('uncle')
uncle(X, Y) <= male(X) & parent(Z, Y) & sibling(Z, X)
# Define aunty/2 predicate
pyDatalog.create_terms('aunty')
aunty(X, Y) <= female(X) & parent(Z, Y) & sibling(Z, X)
# Define sibling
pyDatalog.create_terms('sibling')
sibling(X, Y) <= parent(Z, X) & parent(Z, Y) & (X != Y)
# Query to return the cousin of Adam
print(cousin('Adam', X))
# Query to return the grandfather of Elise
print(grandfather(X, 'Elise'))
```

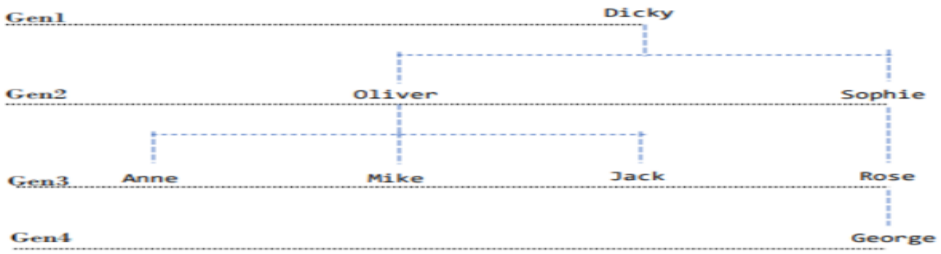
Output:

```
X
-
Lisa
Joar
Elise
```

```
X
-
Lennari
```

Q7.

Implement the following using pyDatalog for the family tree



Males: Dicky, Oliver, Mike, Jack, George

Females: Anne, Rose, Sophia

Define new predicates for the following family relations: Father, Mother, Sister, Brother, Grandmother, Grandfather, Ancestor, Cousin, Uncle, Son and Daughter.

Your program should be able to answer the following question.

a. Was George the parent of Oliver?
b. Who was Oliver's parent?
c. Who were the children of Oliver?
d. Who were the brothers of Anne?
e. Who were the cousins of Rose?

Create any of your own questions (five) and test the answers.

Code:

```
from pyDatalog import pyDatalog

pyDatalog.clear()

# Define male/1 predicate
pyDatalog.create_terms('male')
male('Dicky')
male('Oliver')
male('Mike')
male('Jack')
male('George')

# Define female/1 predicate
pyDatalog.create_terms('female')
```

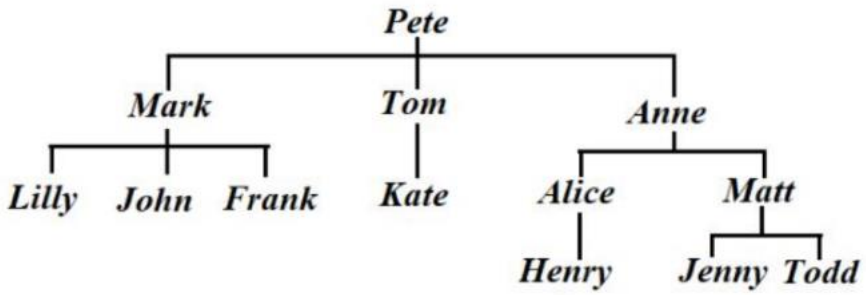
```
female('Anne')
female('Rose')
female('Sophie')

# Define parent/2 predicate
pyDatalog.create_terms('parent')
parent('Dicky', 'Oliver')
parent('Dicky', 'Sophie')
parent('Oliver', 'Anne')
parent('Oliver', 'Mike')
parent('Oliver', 'Jack')
parent('Sophie', 'Rose')
parent('Jack', 'George')

# Define father/2 predicate
pyDatalog.create_terms('father')
father(X, Y) <= male(X) & parent(X, Y)

# Define mother/2 predicate
pyDatalog.create_terms('mother')
mother(X, Y) <= female(X) & parent(X, Y)

# Define sister/2 predicate
pyDatalog.create_terms('sister')
sister(X, Y) <= female(X) & parent(Z, X) & parent(Z, Y) & (X != Y)

# Define brother/2 predicate
pyDatalog.create_terms('brother')
brother(X, Y) <= male(X) & parent(Z, X) & parent(Z, Y) & (X != Y)

# Define grandmother/2 predicate
pyDatalog.create_terms('grandmother')
grandmother(X, Z) <= female(X) & parent(X, Y) & parent(Y, Z)

# Define grandfather/2 predicate
pyDatalog.create_terms('grandfather')
grandfather(X, Y) <= male(X) & parent(X, Z) & parent(Z, Y)

# Define ancestor/2 predicate
pyDatalog.create_terms('ancestor')
ancestor(X, Y) <= parent(X, Y)
ancestor(X, Y) <= parent(X, Z) & ancestor(Z, Y)

# Define cousin/2 predicate
pyDatalog.create_terms('cousin')
cousin(X, Y) <= parent(Z, X) & parent(W, Y) & (Z != W) & sibling(Z, W)

# Define uncle/2 predicate
```

```
pyDatalog.create_terms('uncle')
uncle(X, Y) <= male(X) & parent(Z, Y) & sibling(Z, X)

# Define aunt/2 predicate
pyDatalog.create_terms('aunt')
aunt(X, Y) <= female(X) & parent(Z, Y) & sibling(Z, X)

# Define son/2 predicate
pyDatalog.create_terms('son')
son(X, Y) <= male(X) & parent(Y, X)

# Define daughter/2 predicate
pyDatalog.create_terms('daughter')
daughter(X, Y) <= female(X) & parent(Y, X)

# Query a. Was George the parent of Oliver?
print(parent('George', 'Oliver'))

# Query b. Who was Oliver's parent?
print(parent(X, 'Oliver'))

# Query c. Who were the children of Oliver?
print(parent('Oliver', X))

# Query d. Who were the brothers of Anne?
print(brother(X, 'Anne'))

# Query e. Who were the cousins of Rose?
print(cousin(X, 'Rose'))
```

Output:

```
False
X
--------
Anne
Mike
Jack
cousin
--------
George
```

Q8. Implement the following using pyDatalog for the family tree



Create a simple database containing facts and inference rules. Include facts about family members, such as *male*, *female*, *parent*. Then add rules such as *sister*, *brother*, *sibling*, *father*, *mother*, *grandparent*. Run queries that will answer the following questions about family members: (a) Is Pete Mark's parent? (b) Is Anne Jenny's parent? (c) Who is Todd's father? (d) Who is Tom's sibling? (e) Who is Lilly's brother? (f) Who is Henry's grandparent? (g) Who is Alice's sister? (h) Is Frank Kate's brother? (i) Who is Matt's mother? (j) Is Mark Anne's brother?

Submission file should include your database and screenshots of the results of all queries.

Code:

```
from pyDatalog import pyDatalog

pyDatalog.create_terms('male, female, parent, sister, brother, sibling, father, mother, grandparent')

# Facts about family members
+male('Pete')
+male('Mark')
+male('John')
+male('Frank')
+male('Matt')
+male('Todd')
+female('Lilly')
+female('Kate')
+female('Alice')
+female('Anne')
+female('Jenny')

# Facts about parents and children
+parent('Pete', 'Mark')
+parent('Pete', 'Tom')
+parent('Pete', 'Anne')
+parent('Mark', 'Lilly')
+parent('Mark', 'John')
+parent('Tom', 'Kate')
```

```
+parent('Tom', 'Frank')
+parent('Anne', 'Alice')
+parent('Anne', 'Matt')
+parent('Alice', 'Henry')
+parent('Matt', 'Jenny')
+parent('Todd', 'Jenny')

# Inference rules
sibling(X, Y) <= (parent(Z, X) & parent(Z, Y) & (X != Y))
brother(X, Y) <= (male(X) & sibling(X, Y))
sister(X, Y) <= (female(X) & sibling(X, Y))
father(X, Y) <= (male(X) & parent(X, Y))
mother(X, Y) <= (female(X) & parent(X, Y))
grandparent(X, Y) <= (parent(X, Z) & parent(Z, Y))

# Queries
print(parent('Pete', 'Mark')) # a. Is pete mark’s parent? - Yes
print(parent('Anne', 'Jenny')) # b. Is anne jenny’s parent? - No
print(father(X, 'Todd')) # c. Who is todd’s father? - Tom
print(sibling(X, 'Tom')) # d. Who is tom’s sibling? - Anne
print(brother(X, 'Lilly')) # e. Who is lilly’s brother? - John
print(grandparent(X, 'Henry')) # f. Who is henry’s grandparent? - Pete
print(sister(X, 'Alice')) # g. Who is alice’s sister? - Matt
print(brother('Frank', 'Kate')) # h. Is frank kate’s brother? - Yes
print(mother(X, 'Matt')) # i. Who is matt’s mother? - Anne
print(brother('Mark', 'Anne')) # j. Is mark anne’s brother? - No
```

Output:

```
(True,)
()
X
==
Tom
X
==
John
X
==
Pete
X
==
Matt
(True,)
(True,)
(False,)
```