1. 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: a /
   b c / \ | d e f What are the answers generated by your definitions for the queries: brother(X,Y) cousin(X,Y) grandson(X,Y) descendent(X,Y)

```
!pip install pyDatalog
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyDatalog
  Downloading pyDatalog-0.17.4-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 22.8 MB/s eta 0:00:00
Installing collected packages: pyDatalog
Successfully installed pyDatalog-0.17.4
```

```
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)'))
```

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

2. 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.

```
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))
```

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

3. 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.

```
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))
```

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

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

```
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')
```

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

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

```
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)
```

```
Enter any number:10
N
-------
3628800
```

6. 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

```
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'))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-14-1c5a43aa32ea> in <cell line: 56>()
     54 # Define cousin/2 predicate
     55 pyDatalog.create_terms('cousin')
---> 56 cousin(X, Y) <= parent(Z, X) & parent(W, Y) & (Z != W) & sibling(Z, W)
     57
     58 # Define grandfather/2 predicate

NameError: name 'W' is not defined
```

SEARCH STACK OVERFLOW

7. Implement the following using pyDatalog for the family tree

```
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')x
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'))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-bc8ec98fa531> in <cell line: 60>()
     58 # Define cousin/2 predicate
     59 pyDatalog.create_terms('cousin')
---> 60 cousin(X, Y) <= parent(Z, X) & parent(W, Y) & (Z != W) & sibling(Z, W)
     61
     62 # Define uncle/2 predicate

NameError: name 'W' is not defined
```

SEARCH STACK OVERFLOW

8. Implement the following using pyDatalog for the family tree

```
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'))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-21c2aa9d3357> in <cell line: 59>()
     57 # Define cousin/2 predicate
     58 pyDatalog.create_terms('cousin')
---> 59 cousin(X, Y) <= parent(Z, X) & parent(W, Y) & (Z != W) & sibling(Z, W)
     60
     61 # Define uncle/2 predicate

NameError: name 'W' is not defined
```

SEARCH STACK OVERFLOW