DATE

EXPT. NO. 1

PAGE NO.

Execute the Basic Operations of SWI Prolog and python along with the installation process of python, Jupyter Notebook and SWI Prolog.

Official Website : https://www.swi-prolog.org/

Prolog version: 8.4.3

Installation Process of SWI Prolog :—

i) Search SWI Prolog download in browser.

ii) Click on SWI-Prolog official website and go to 'Downloads' section, after which click on 'stable'.

iii) Select the suitable os and bit size.

iv) Select the checkboxes to get the actual link and download.

v) Allow access to run the application.

vi) In set-up box, tap 'next' to continue and then 'I agree'.

vii) Add swipl to system path for all user and create desktop icon (optional).

viii) Select the folder, path or default and then 'Next'.

ix) Click on next and 'install'.

x) After few minutes, the SWI Prolog will be installed and ready to run.

PIONEER©

DATE

EXPT. NO.

pioneerpaper in
PAGE NO.

Official Website : https://www.python.org/

Python version : 3.10.8

Installation Process of Python Jupyter Notebook :-

i) Firstly search python download in browser.

ii) Click on the official website page for downloading Python.

iii) Click on the button 'Download Python' to download and install.

iv) Next open command prompt or terminal to install Jupyter Notebook.

v) Install the Jupyter Notebook with :

```
pip install notebook
```

vi) Once installed launch it with :

```
jupyter notebook
```

vii) It will open a localhost in browser and is ready to work with.

PIONEER®

DATE

EXPT. NO.

pioneerpaper in
PAGE NO.

Basic Operations in SWI Prolog :—

The basic operation is arithmetic operation.

Arithmetic operations :

```
? - X is 10.5 + 4.7 * 2
X = 19.1
? - Y is 10, Z is y+1
Z = 11
? - X is sqrt(36)
X = 6
? - x is 10, y is -x- 2
x = 10
Y = -12
```

Basic Operations in Python :—

The basic operations in Python are :—

i) Arithmetic - addition (a+b), substraction (a-b), division (a/b), multiplication (a * b), modulus (a % b) etc.

ii) Assignment - =, +=, -=, */, etc

iii) Comparison - ==, !=, etc.

iv) logical - and, or, not

v) Identify - is, is not

vi) Membership - in, not in.

PIONEER®

Output for Program (1) :-

```
?- dog(x).
x - puppy ;
x - kutty ;
x - jimmy.
?- cat(x).
x - valu ;
x - miaw ;
x - mouse.
?- animal(y).
y - puppy ;
y - kutty ;
y - jimmy ;
y - valu ;
y - miaw ;
y - mouse.
```

Output for Program (2) :-

```
?- likes(ram, x).
x - mango.
?- likes(ram, mango).
← true
?- own(john, silver).
  false
```

DATE

EXPT. NO
2, 3

PAGE NO

Implementation of Relational Tree Structure in SWI Prolog.

Algorithm / Flowchart :-

i) Check whether the relation is true based on the facts given.

ii) If true, return answer for the question else if false terminate the process.

Program Listing (1) :-

```
dog (puppy).
dog (kutty).
dog (jimmy).
cat (valu).
cat (miaw).
cat (mouse).
animal (Y) :- dog (Y); cat (Y).
```

Program Listing (2) :-

```
likes (ram, mango).
girl (seema).
red (rose).
likes (bill, candy).
owns (john, gold).
```

PIONEER®

Output:

```
?- circuit (0,1,1,1).
 ON
 true
?- circuit (0,0,0,0).
 OFF
 false
?- circuit (1, 0,0, 0).
 OFF
true
```

DATE

EXPT. NO. 4

PAGE NO. 6

Implementation of Circuit

Algorithm / Flowchart:

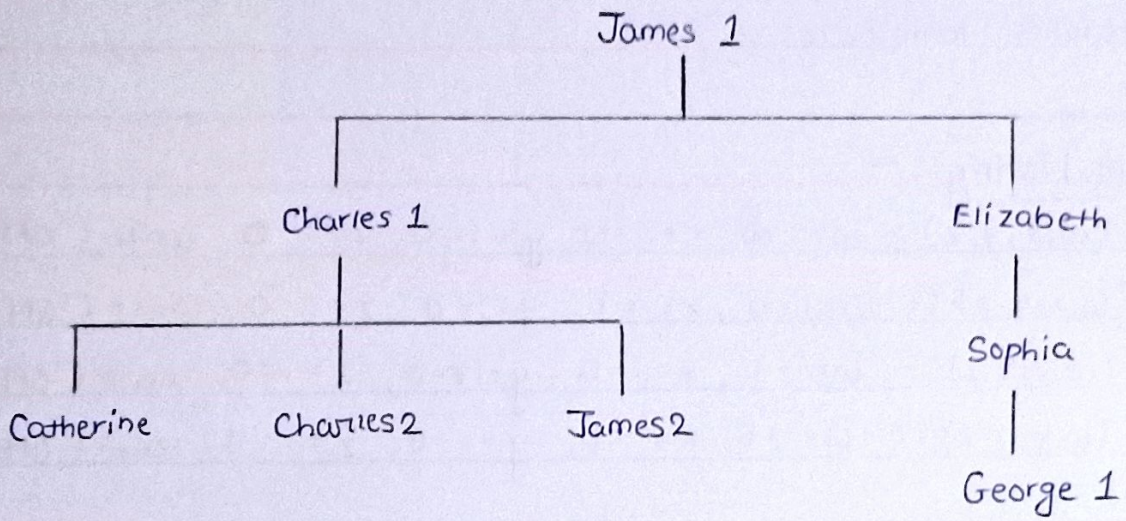
i) Check facts whether the light bulb is on or off.
ii) If the query is true, check whether the light is on or off.
iii) Otherwise terminate

Programme Listing:

```
circuit(w,x,y,z):- w=:=0, x=:=0, y=:=0, z=:=0, write('OFF').
circuit(w,x,y,z):- w=:=0, x=:=1, y=:=0, z=:=0, write('OFF'),
circuit(w,x,y,z):- w=:=1, x=:=0, y=:=0, z=:=0, write('OFF').
circuit(w,x,y,z):- w=:=1, x=:=1, y=:=0, z=:=0, write('OFF').

circuit(w,x,y,z):- w=:=0, x=:=0, y=:=1, z=:=0, write('OFF').
circuit(w,x,y,z):- w=:=0, x=:=1, y=:=1, z=:=0, write('OFF').
circuit(w,x,y,z):- w=:=1, x=:=0, y=:=1, z=:=0, write('OFF').
circuit(w,x,y,z):- w=:=1, x=:=1, y=:=1, z=:=0, write('OFF').

circuit(w,x,y,z):- w=:=0, x=:=1, y=:=1, z=:=1, write('ON').
circuit(w,x,y,z):- w=:=1, x=:=0, y=:=1, z=:=1, write('ON').
circuit(w,x,y,z):- w=:=1, x=:=1, y=:=0, z=:=1, write('ON').
circuit(w,x,y,z):- w=:=1, x=:=1, y=:=1, z=:=1, write('ON').
```


James 1
Charles 1
Elizabeth
Catherine
Charles 2
James 2
Sophia
George 1

DATE

EXPT. NO. 5

pioneerpaper.in PAGE NO.

Implementation of Predecessors & Successors in SWI Prolog.

Algorithm / Flowchart :-

i) Run the facts, if true, check the question, then return name otherwise return just true.

ii) Otherwise terminate the process.

Program Listing :-

```
male(james1).
male(james2).
male(charles1).
male(charles2).
male(george1).
female(catherine).
female(elizabeth).
female(sophia).
parent(sofia, elizabeth).
parent(george1, sophia).
parent(elizabeth, james1).
Parent(charles1, james1).
paren+(chatherine, charles1).
parent(charles2, charles1).
parent(james2, charles1).
son(X,Y) :-male(Y), parent(Y,X).
daughter(X,Y) :- female(Y), parent(Y,X).
```

PIONEER®

Output :-

```
?- male(james1).
   True.
?- female(sophia).
   True
?- son(james1, charles1).
   True
?- son(james1, charles2).
   False.
```

DATE

EXPT. NO.

pioneerpaper.in
PAGE NO.

father(x,y) :- male(x), parent(x,y).

mother(x,y) :- female(y), parent(x,y).

brother(x,y) :- male(y), parent(x,z), parent(y,z).

sister(x,y) :- female(y), parent(x,z), parent(y,z).

grandmother(x,z) :- female(z), parent(x,y), parent(y,z).

grandfather(x,z) :- male(z), parent(x,y), parent(y,z).

grandson(x,z) :- son(x,y), son(y,z);
daughter(x,y), son(y,z).

grand_daughter(x,z) :- daughter(x,y), daughter(y,z);
son(x,y), daughter(y,z).

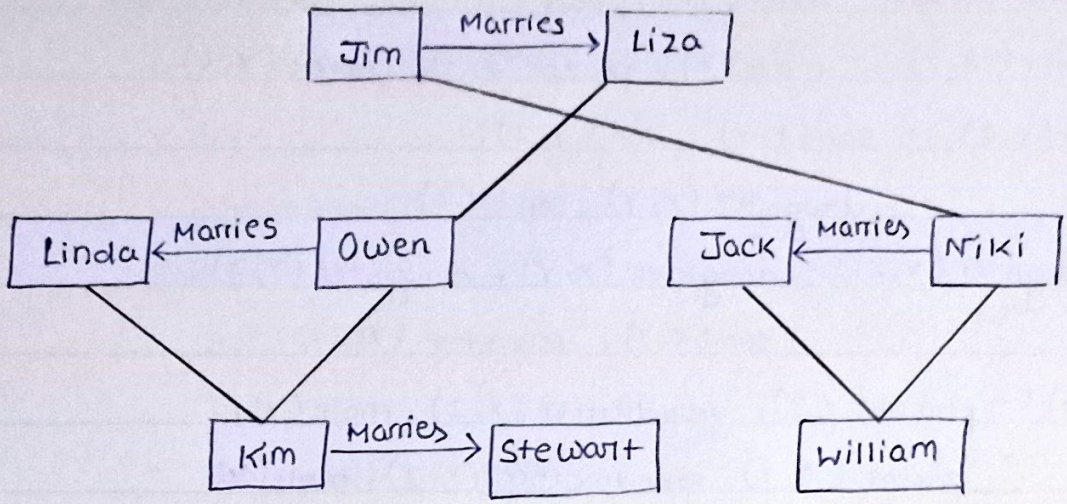
uncle(x,y) :- parent(y,z), grandfather(x,z), male(y);
parent(y,z), grandmother(x,z), male(y).

aunt(x,y) :- parent(y,z), grandfather(x,z), female(y);
parent(y,z), grandmother(x,z), female(y).

nephew(x,y) :- brother(z,x), son(z,y);
sister(z,x), son(z,y).

niece(x,y) :- brother(z,x), daughter(z,y);
sister(z,x), daughter(z,y).

PIONEER®


Jim
Marries
Liza
Linda
Marries
Owen
Jack
Marries
Niki
Kim
Marries
Stewart
William

DATE

DATE

EXPT. NO.
6

pioneerpaper.in
PAGE NO.

Implementation of Predecessors and Successors in SWI Prolog.

Algorithm / Flowchart :-

i) Run the facts, if true, check the question, then return name otherwise return just true.

ii) Otherwise terminate the process.

Program Listing :-

```
male(jim).
male(owen).
male(jack).
male(stewart).
male(william).
female(liza).
female(linda).
female(niki).
female(kim).
parent(owen, liza).
parent(owen, jim).
parent(niki, jim).
parent(niki, liza).
parent(kim, linda).
parent(kim, owen).
parent(william, jack).
parent(william, niki).
```

PIONEER®

Output :-

?- father (owen, jim).

true.

? parent(owen, x).

x = jim

x = liza.

?- female (owen).

false.

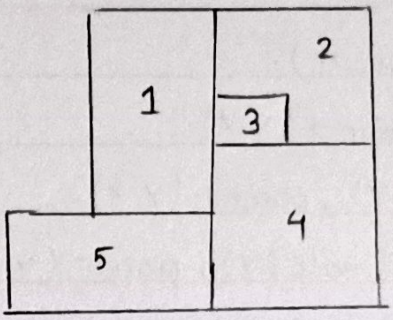
DATE

EXPT. NO.

pioneerpaper.in
PAGE NO.

```
husband (liza, jim).
husband (lind, owen).
husband (niki, jack).
husband (kim, stewart).
wife (x, y) :- husband (y, x).
son (x, y) :- male (y), parent (y, x).
daughter (x, y) :- female (y), parent (y, x).
father (x, y) :- male (y), parent (x, y).
mother (x, y) :- female (y), parent (x, y).
brother (x, y) :- male (y), parent (x, z), parent (y, z), x \== y.
sister (x, y) :- female (y), parent (x, z), parent (y, z), x \== y.
grandmother (x, z) :- female (z), parent (x, y), parent (y, z).
grandfather (x, z) :- male (z), parent (x, y), parent (y, z).
grandson (x, z) :- son (x, y), son (y, z);
                   daughter (x, y), son (y, z).
grand_daughter (x, z) :- daughter (x, y), daughter (y, z);
                         son (x, y), daughter (y, z).
uncle (x, y) :- parent (x, z), brother (z, y);
                parent (x, z), sister (z, w), husband (w, y).
aunt (x, y) :- parent (x, z), sister (z, y);
               uncle (x, z), wife (z, y).
```

PIONEER®



Output:-

?- conflict(1,2).

false

?- conflict(1,4).

false

DATE

EXPT. NO. 7

PAGE NO.

Implementation of graph colouring in SWI Prolog :-

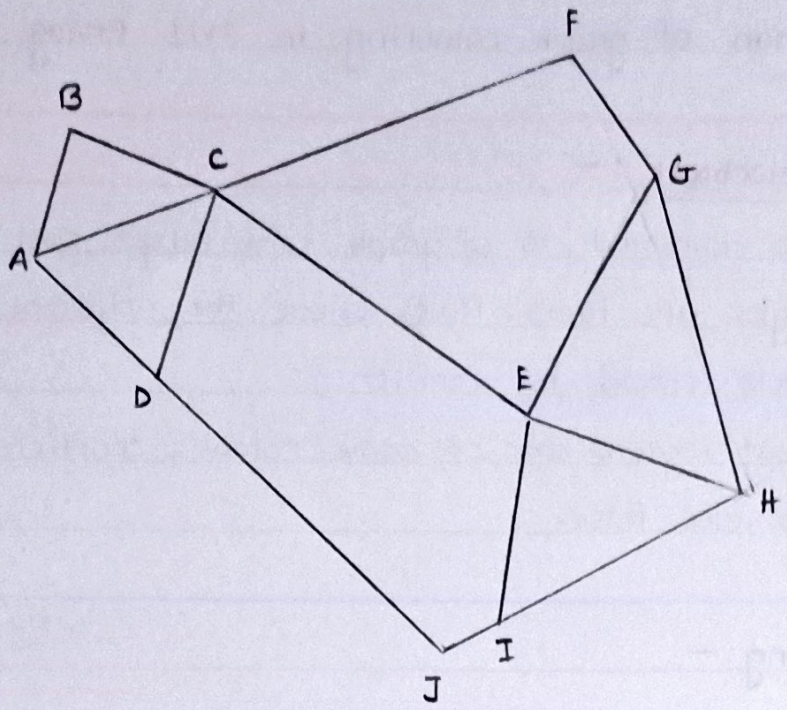
Algorithm/Flowchart :-

i) Regions are connected to a graph with edges and vertices, where edges are facts. Rules check the adjacent regions. Another rule checks for conflict.

ii) If adjacent regions are of same colour, conflict occours, return true, else false.

Program Listing :-

```
adjreg (x, y) :- edge (x, y), edge (y, x).
colour (1, green).
colour (2, blue).
colour (3, yellow).
colour (4, pink).
colour (5, blue).
conflict (x, y) :- colour (x, c1), colour (y, c2), c1 = c2.
```



Output :-

```
?- ad(d,X).
   X = a
   X = c
   X = j
?- ad(f,X).
   X = c
   X = g
?- ad(a,i).
   false
?- ad(a,c).
   true.
```

DATE

EXPT. NO. 8

PAGE NO.

Adjacent and Non-Adjacent vertex.

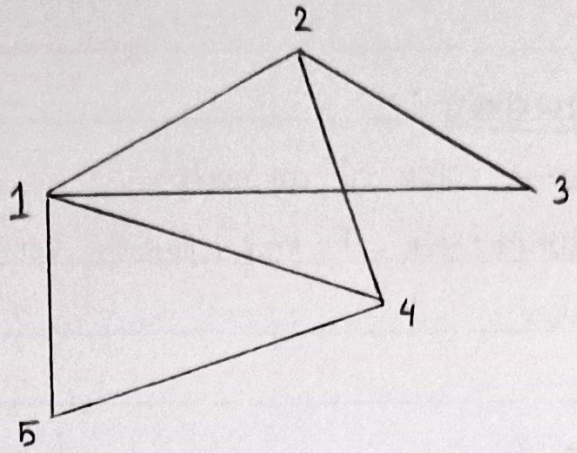
Algorithm / Flowchart :-

i) Find adjacent value of all nodes.

ii) Check if connected, if true, return vertex, otherwise terminate the process.

Program Listing :-

```
ad (a,b).
ad(a,c).
ad(a,d).
ad(b,a).
ad(b,c).
ad(c,b).
ad(c,a).
ad(c,f).
ad(c,d).
ad(c,e).
ad(d,a).
ad(d,c).
ad(d,j).
ad(e,c).
ad(e,g).
ad(e,i).
ad(e,h).
ad(g,e).
ad(g,f).
ad(g,h).
ad(h,g).
ad(h,e).
ad(h,i).
ad(i,j).
ad(i,h).
ad(i,e).
ad(j,d).
ad(j,i).
```



Output :-

```
?- adjreg (1, X).
   X = 2
   X = 3
   X = 4
   X = 5
```

DATE

EXPT. NO. 9

pioneerpaper.in PAGE NO.

Adjacent and Non-Adjacent Regions:-

Algorithm / Flowchart :-

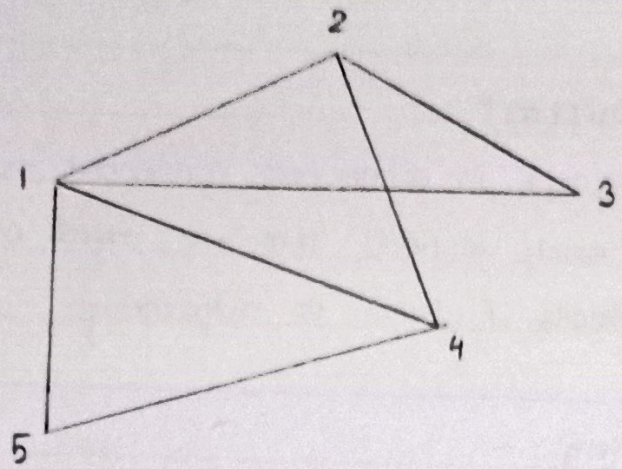
i) Regions of graph is connected converted to graph that consists of edges and vertices. Then the edges are implemented as facts.

ii) A rule checks if there is adjacency.

Program Listing :-

```
adjreg (X,Y) :- edge (X,Y); edge (Y,X).
edge (1,2).
edge (1,5).
edge (1,4).
edge (1,3).
edge (2,3).
edge (2,4).
edge (4,5).
```

PIONEER®



Output :-

```
?- edge (1, X, Y).
    X = 2 , Y = e1
    X = 5 , Y = e2
    X = 3 , Y = e5
    X = 4 , Y = e6
?- edge ( 1, 2, e1).
    true
?- edge ( 1, 2, e3).
    False.
```

DATE

EXPT. NO. 10

pioneerpaper.in
PAGE NO.

Adjacent and Non-Adjacent Edges

Algorithm / Flowchart :-

i) Run predicates, if it is an edge, check the question. If true, return true and print edges.

ii) Otherwise terminate the process.

Program Listing :-

```
edge (1,2,e1).
edge (1,5,e2).
edge (1,3,e5).
edge (1,4,e6).
edge (2,3,e3).
edge (2,4,e4).
edge (3,2,e3).
edge (3,1,e5).
edge (4,1,e6).
edge (4,5,e7).
edge (5,1,e2).
edge (5,4,e7).
```

PIONEER®

DATE

EXPT. NO. 11

pioneerpaper.in
PAGE NO.

Write a program in SWI Prolog where the facts are given as:

Jack owns BMW car.

John owns chevy car.

Olivia owns civic car.

Jane owns chevy car.

BMW car is sedan.

Civic car is sedan

Chevy car is truck

To satisfy the list of queries:-

What does John own?

Does John own something?

Who owns car chevy?

Does Jane own sedan?

Does Jane own truck?

Algorithm/Flowchart :-

i) Find what type of cars are available.

ii) Person Names.

iii) Type of the car.

iv) Who owns what type of car.

v) Who owns what car with type.

PIONEER®

Output :-

```
?- owns (john, X).
   X = chevy.
?- owns (X, chevy).
   X = John
   X = Jane
?- cartype (bmw, X).
   X = sedan
?- person (civic).
   false
?- cartype (X, sedan).
   X = bmw
   X = civic
```

DATE

EXPT. NO.

PAGE NO.

Program Listing :-

```
person (jane).
person (jack).
person (john).
person (olivia).
car (bmw).
car (chevy).
car (civic).

cartype (bmw, sedan).
cartype (chevy, truck).
cartype (civic, sedan).

owns (Jack, bmw).
owns (John, chevy).
owns (olivia, civic).
owns (Jane, chevy).
```

PIONEER®

Output :-

cat (X) → X = fubby

spot (fubby, X) → X = black

spot (figaro, X) → X = white

owns (manny, fubby) → True

loves (mary, Y) → Y = Fubby.

DATE

EXPT. NO. 12

PAGE NO.

Facts : Fubby is a cat. Fubby has black spots. Figaro is a dog. Figaro has white spots.

Rules : Mary owns a pet if it is a cat and it has black spots. If someone owns something he loves it.

Implement the rule-base in SWI-prolog.

Algorithm / Flowchart :—

i) Identify fubby as a cat and figaro as a dog.

ii) Fubby has black spots and Figaro has white spots.

iii) Identify Mary owns a cat or a dog.

iv) Check if its a cat and it has black spots.

v) If Mary owns that, she loves it.

Program listing :—

```
cat(fubby).
dog(figaro).
spot(fubby, black).
spot(figaro, white).
owns(mary, Y) :- cat(Y), spot(Y,Z), Z = black.
loves(X, Y) :- owns(X, Y).
```