Experiment-1

Aim:- Introduction to Prolog.

Application:-

Prolog is a declarative programming language widely used for tasks that involve symbolic reasoning, logic-based problem-solving, and artificial intelligence (AI). Its unique strengths lie in its ability to handle complex relationships, backtracking, and pattern matching efficiently. Below are some key **applications of Prolog**:

- Artificial Intelligence (AI) and Expert Systems
- Natural Language Processing (NLP)
- Knowledge Representation and Reasoning
- Solving Puzzles and Games
- Constraint Logic Programming (CLP)

Syntax:-

Prolog is a declarative programming language that uses facts, rules, and queries to represent and solve problems. Below is a breakdown of the key components of Prolog syntax:

Facts

Facts represent basic information or knowledge that is always true. A fact consists of a head (predicate) followed by an argument.

Eg: likes(john, pizza).

Rules

Rules are used to define relationships between facts. A rule has a head (what we want to prove or deduce) and a body (conditions that must be true for the head to be true). Rules are written with a `:-` symbol, where the body precedes the head.

Eg: grandfather(X, Y) :- father(X, Z), father(Z, Y).

Queries

Queries are used to ask Prolog questions about the facts and rules that have been defined. A query is similar to a fact, but it is not terminated with a period when being asked. When asking a query, Prolog will attempt to match it with facts and rules.

Eg: ?- likes(john, pizza).

Variables

Variables in Prolog are written in **uppercase** letters or with an underscore **_**. Variables can stand for any term and are used to match facts or rules.

Eg: likes(john, X).

Atoms

Atoms are constant values in Prolog. These can be:

- **Simple atoms**: Lowercase letters, or atoms enclosed in single quotes if they contain spaces or special characters.
- **Examples**:
  - `john`
  - `'big apple'`
  - `house`

Compound Terms

A compound term consists of a functor and its arguments. The functor is similar to a function name, and the arguments are the terms associated with it. Arguments can be atoms, numbers, variables, or other compound terms.

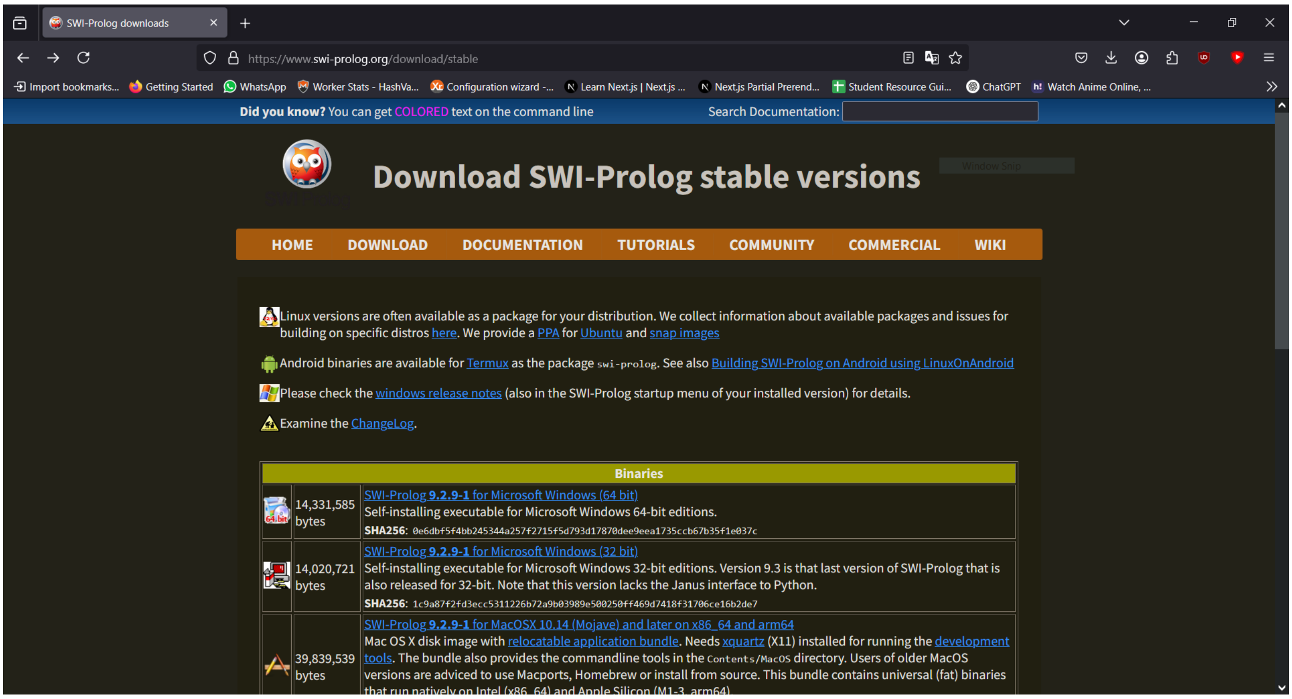
Eg: parent(john, mary).
book('Prolog for Beginners', john).

Software:-

SWI-Prolog is a widely-used open-source implementation of the Prolog programming language. It provides a comprehensive environment for developing logic-based applications. SWI-Prolog is known for its powerful and efficient execution engine, rich set of libraries, and active community.

Installation of SWI-Prolog:-

- Go to the official SWI-Prolog download page: https://www.swi-prolog.org/download/stable



Did you know? You can get COLORED text on the command line

Download SWI-Prolog stable versions

Linux versions are often available as a package for your distribution. We collect information about available packages and issues for building on specific distros here. We provide a PPA for Ubuntu and snap images

Android binaries are available for Termux as the package `swi-prolog`. See also Building SWI-Prolog on Android using LinuxOnAndroid

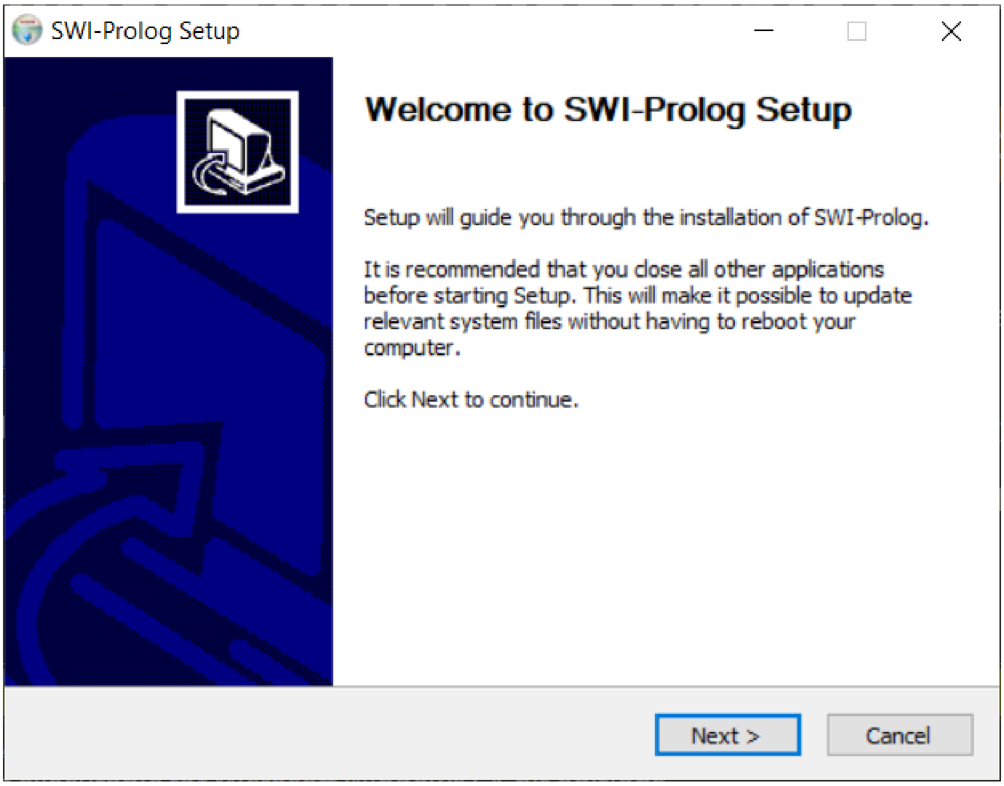
Please check the windows release notes (also in the SWI-Prolog startup menu of your installed version) for details.

Examine the ChangeLog.

Binaries		
	14,331,585 bytes	SWI-Prolog **9.2.9-1** for Microsoft Windows (64 bit) Self-installing executable for Microsoft Windows 64-bit editions. **SHA256**: `0e6dbf5f4bb245344a257f2715f5d793d17870dee9eea1735ccb67b35f1e037c`
	14,020,721 bytes	SWI-Prolog **9.2.9-1** for Microsoft Windows (32 bit) Self-installing executable for Microsoft Windows 32-bit editions. Version 9.3 is that last version of SWI-Prolog that is also released for 32-bit. Note that this version lacks the Janus interface to Python. **SHA256**: `1c9a87f2fd3ecc5311226b72a9b03989e500250ff469d7418f31706ce16b2de7`
	39,839,539 bytes	SWI-Prolog **9.2.9-1** for MacOSX 10.14 (Mojave) and later on x86_64 and arm64 Mac OS X disk image with relocatable application bundle. Needs xquartz (X11) installed for running the development tools. The bundle also provides the commandline tools in the `Contents/MacOS` directory. Users of older MacOS versions are adviced to use Macports, Homebrew or install from source. This bundle contains universal (fat) binaries that run natively on Intel (x86_64) and Apple Silicon (M1-3, arm64).

1. Download the installer for Windows.

2. Run the installer and follow the installation instructions.
3. Click Next.



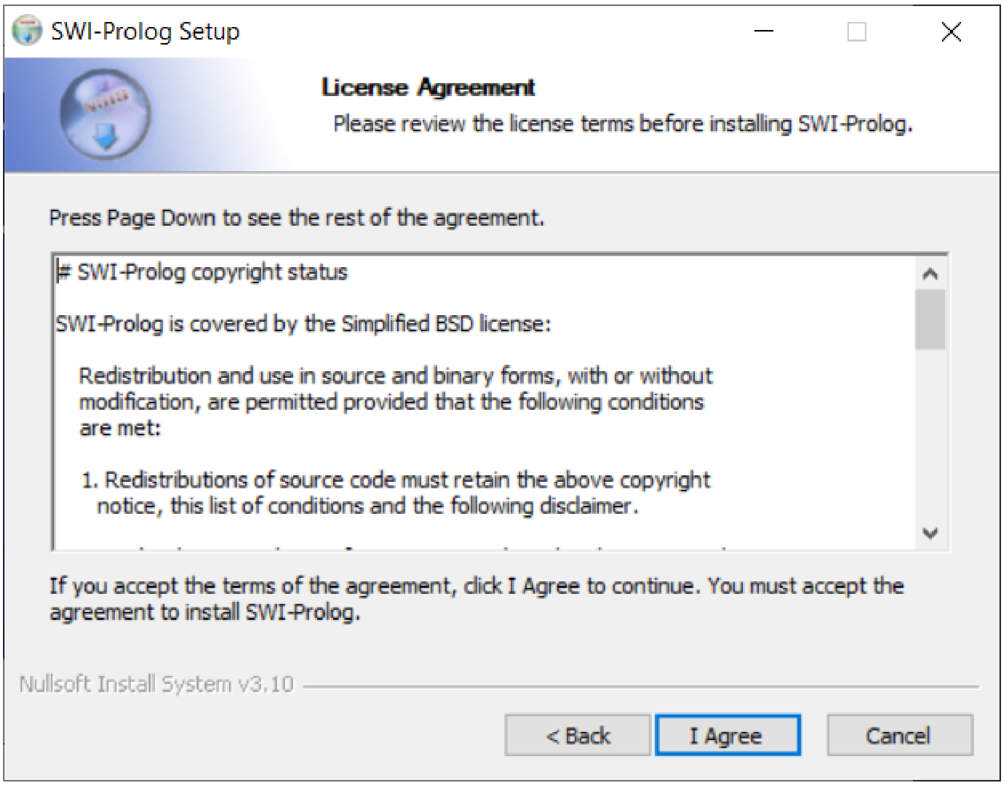
Welcome to SWI-Prolog Setup

Setup will guide you through the installation of SWI-Prolog.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

4. Click On ‘I Agree’.



License Agreement

Please review the license terms before installing SWI-Prolog.

Press Page Down to see the rest of the agreement.

SWI-Prolog copyright status

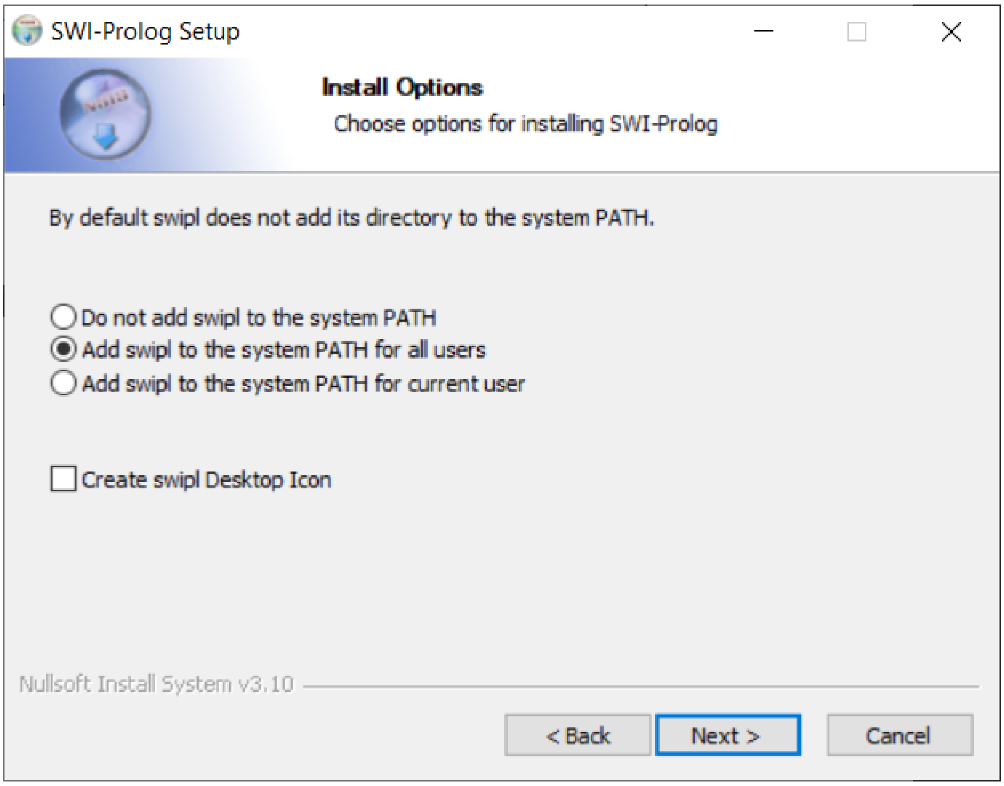
SWI-Prolog is covered by the Simplified BSD license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install SWI-Prolog.

5. Select 2nd Radio button and click next.



Install Options

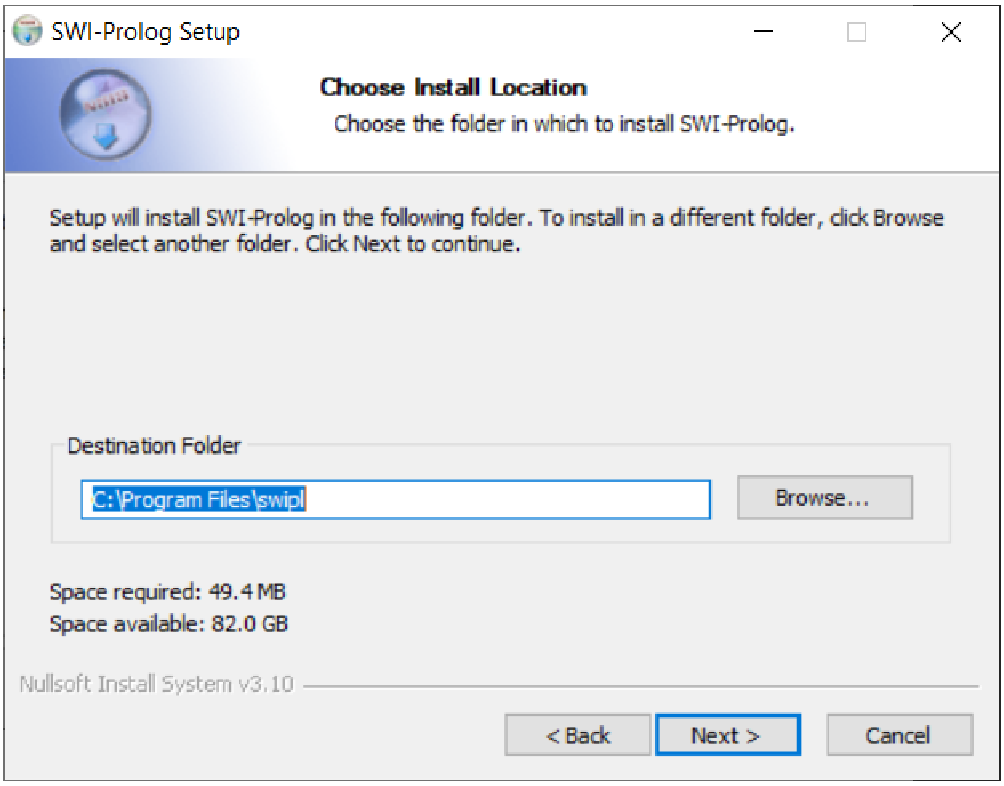
Choose options for installing SWI-Prolog

By default swipl does not add its directory to the system PATH.

○ Do not add swipl to the system PATH
● Add swipl to the system PATH for all users
○ Add swipl to the system PATH for current user

□ Create swipl Desktop Icon

6. Click Next.



Choose Install Location

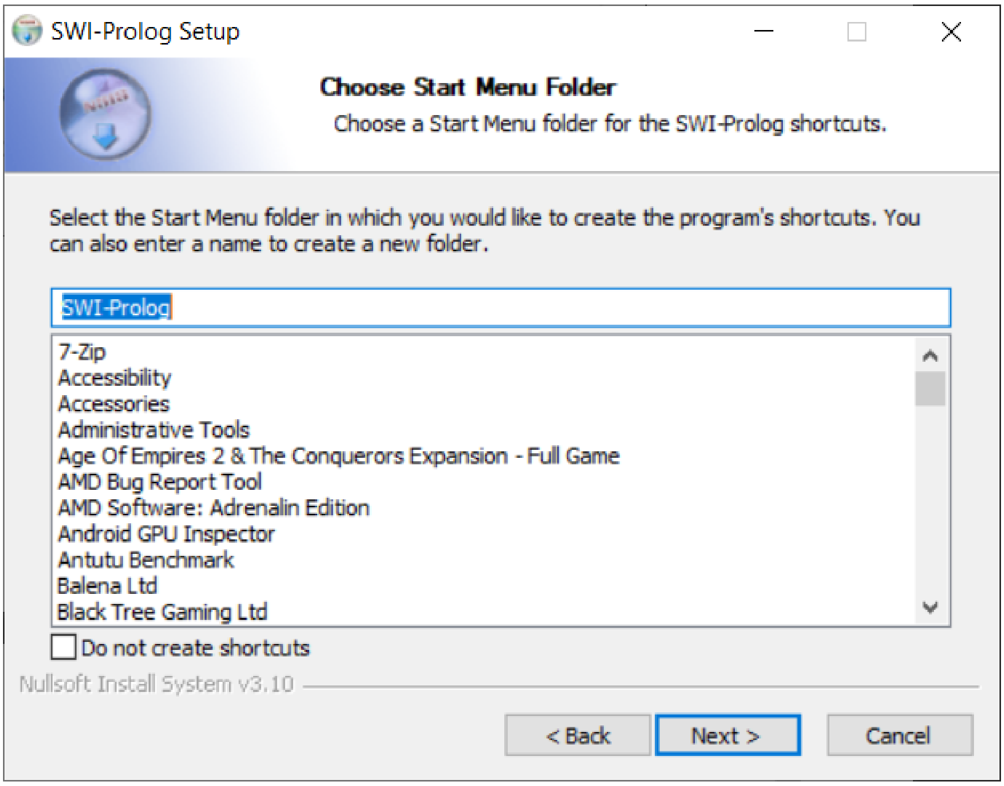
Choose the folder in which to install SWI-Prolog.

Setup will install SWI-Prolog in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files\swipl

Space required: 49.4 MB
Space available: 82.0 GB

7. Click Next.



SWI-Prolog Setup

Choose Start Menu Folder

Choose a Start Menu folder for the SWI-Prolog shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

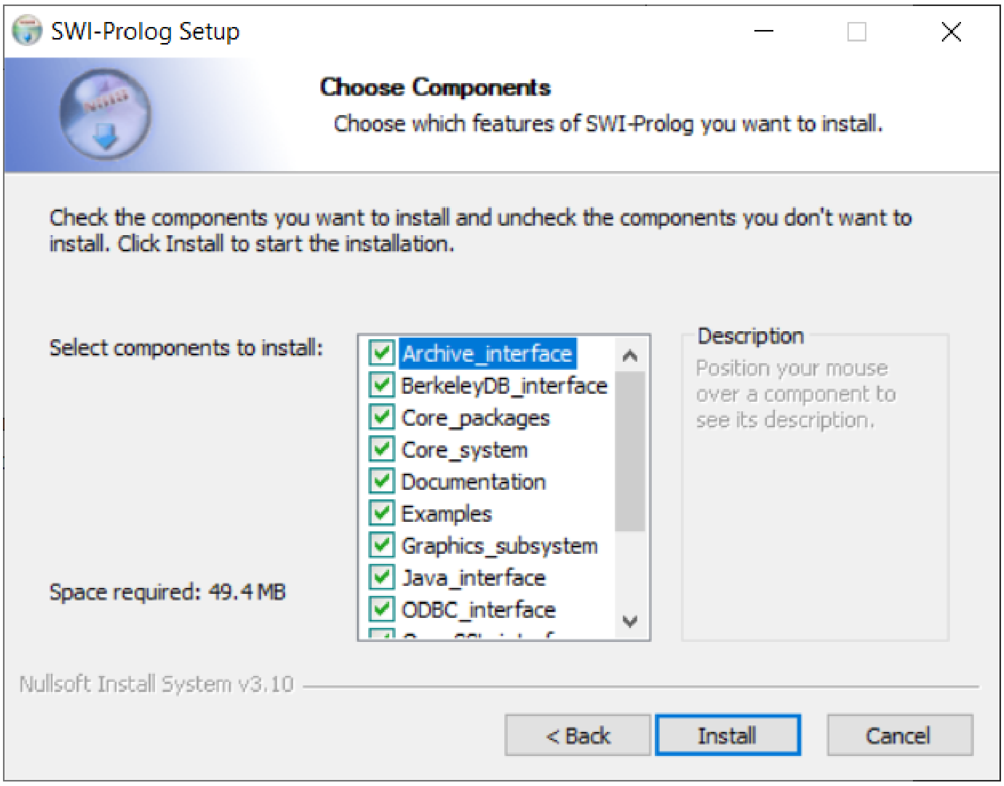
SWI-Prolog

7-Zip
Accessibility
Accessories
Administrative Tools
Age Of Empires 2 & The Conquerors Expansion - Full Game
AMD Bug Report Tool
AMD Software: Adrenalin Edition
Android GPU Inspector
Antutu Benchmark
Balena Ltd
Black Tree Gaming Ltd

□ Do not create shortcuts

Nullsoft Install System v3.10

8. Click Install.



SWI-Prolog Setup

Choose Components

Choose which features of SWI-Prolog you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Install to start the installation.

Select components to install:

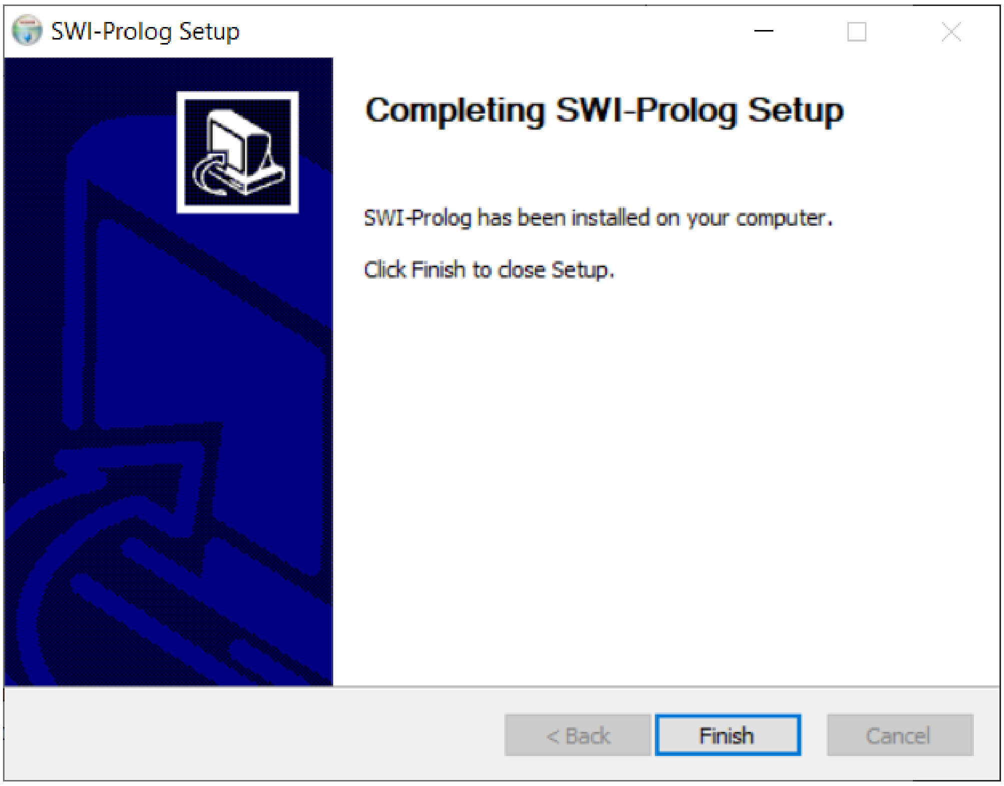
☑ Archive_interface
☑ BerkeleyDB_interface
☑ Core_packages
☑ Core_system
☑ Documentation
☑ Examples
☑ Graphics_subsystem
☑ Java_interface
☑ ODBC_interface

Space required: 49.4 MB

Description

Position your mouse over a component to see its description.

Nullsoft Install System v3.10

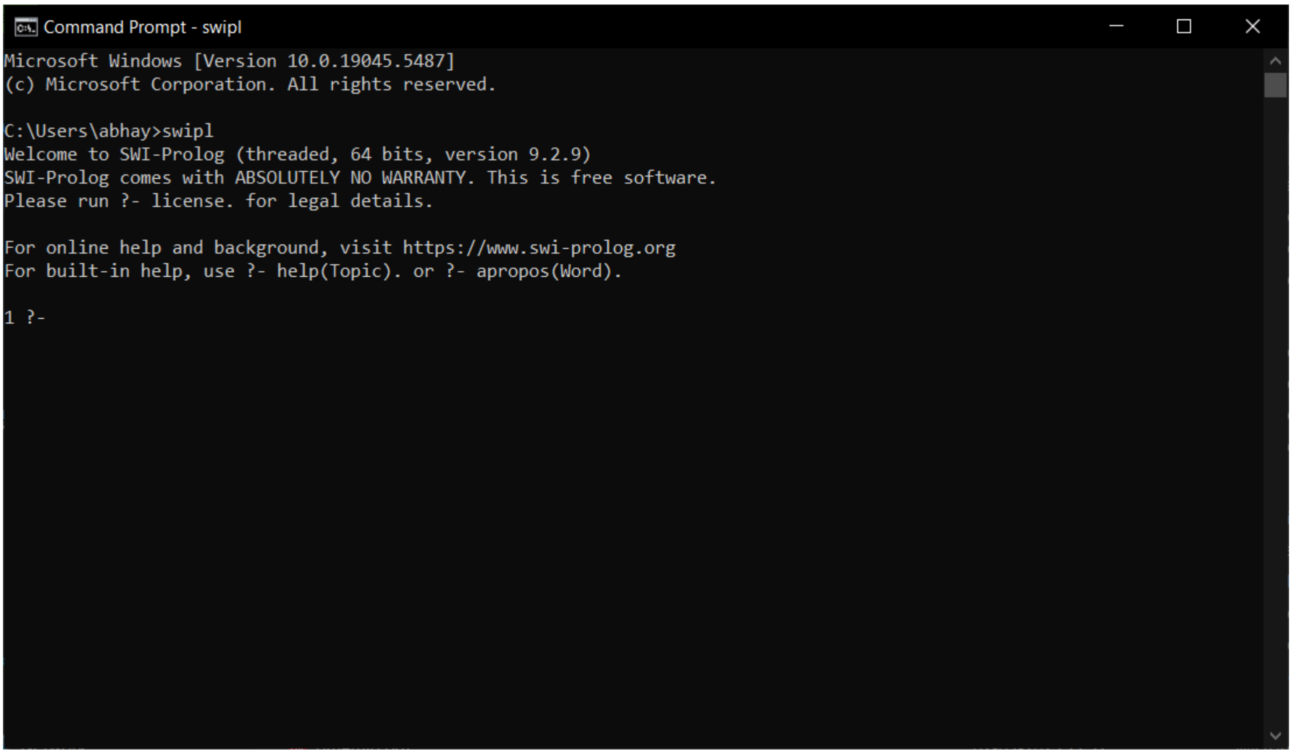


Completing SWI-Prolog Setup

SWI-Prolog has been installed on your computer.

Click Finish to close Setup.

9. Once installed, you can run SWI-Prolog by clicking the SWI-Prolog icon or using the command prompt by typing `swipl`.



```
Microsoft Windows [Version 10.0.19045.5487]
(c) Microsoft Corporation. All rights reserved.

C:\Users\abhay>swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
```

Experiment – 2

Aim:- Write a program to implement the input-output or predicate of PROLOG.

a. Calculate the square of a number.

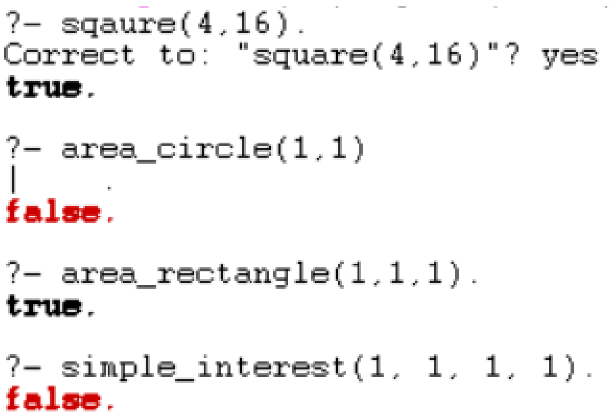
b. Calculate the area of a circle, square, and rectangle.

c. Calculate the simple interest.

Code:-

```
% Calculate the square of a number
square(Number, Result) :-
  Result is Number * Number.

% Calculate the area of a circle
area_circle(Radius, Area) :-
  Area is 3.14159 * Radius * Radius.

% Calculate the area of a square
area_square(Side, Area) :-
  Area is Side * Side.

% Calculate the area of a rectangle
area_rectangle(Length, Width, Area) :-
  Area is Length * Width.

% Calculate the simple interest
simple_interest(Principal, Rate, Time, Interest) :-
  Interest is (Principal * Rate * Time) / 100.
```

Output:-



```
?- sqaure(4,16).
Correct to: "square(4,16)"? yes
true.

?- area_circle(1,1)
|    .
false.

?- area_rectangle(1,1,1).
true.

?- simple_interest(1, 1, 1, 1).
false.
```

Experiment – 3

Aim:- Write a program to implement local variables and conditional statements in PROLOG.

a. To check whether a number is even or odd.

b. To find the maximum of two numbers.

c. To find grades of students based on marks achieved:

● Marks ≥ 90 (A Grade).

● 75 ≤ Marks ≤ 90 (B Grade).

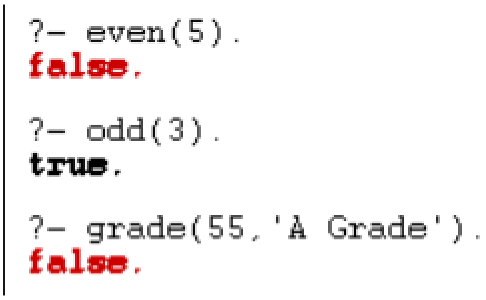
● 50 ≤ Marks ≤ 75 (C Grade).

● Marks < 50 (Fail).

Code:-

```
% Check whether a number is even or odd
even(Number) :-
  Number mod 2 =:= 0.

odd(Number) :-
  Number mod 2 =\= 0.

% Find the maximum of two numbers
max(X, Y, X) :-
  X >= Y.
max(X, Y, Y) :-
  Y > X.

% Find grades of students based on marks achieved
grade(Marks, 'A Grade') :-
  Marks >= 90.
grade(Marks, 'B Grade') :-
  Marks >= 75,
  Marks < 90.
grade(Marks, 'C Grade') :-
  Marks >= 50,
  Marks < 75.
grade(Marks, 'Fail') :-
  Marks < 50.
```

Output:-



```
?- even(5).
false.

?- odd(3).
true.

?- grade(55,'A Grade').
false.
```

Experiment – 4

Aim:- Write simple facts for the statements using PROLOG

a. Ram likes mango.

b. Seema is a girl.

c. Bill likes Cindy.

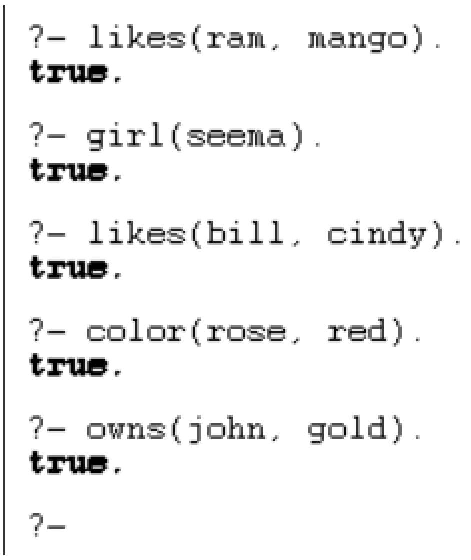
d. Rose is red.

e. John owns gold.

Code:-

```
% Facts
likes(ram, mango).
girl(seema).
likes(bill, cindy).
color(rose, red).
owns(john, gold).
```

Output:-



```
?- likes(ram, mango).
true.

?- girl(seema).
true.

?- likes(bill, cindy).
true.

?- color(rose, red).
true.

?- owns(john, gold).
true.

?-
```

Experiment – 5

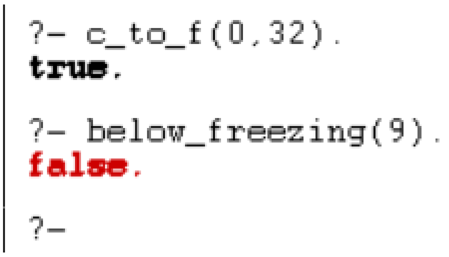
Aim:- Write predicates: one converts centigrade temperatures to Fahrenheit, and

the other checks if a temperature is below freezing using PROLOG.

Code:-

```
%Write predicates, one converts centigrade temperatures to Fahrenheit, the other
checksif a temperature is  below freezing using PROLOG.
% Convert Centigrade to Fahrenheit
c_to_f(C, F) :-
  F is (C * 9 / 5) + 32.

% Check if temperature is below freezing
below_freezing(C) :-
  C < 0.
```

Output:-



```
?- c_to_f(0,32).
true.

?- below_freezing(9).
false.

?-
```

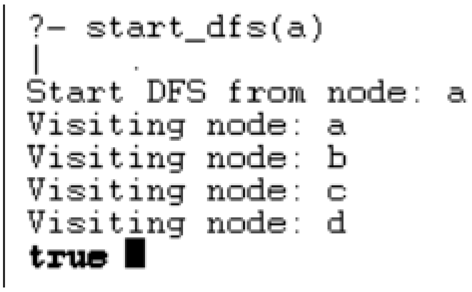
Experiment – 6

Aim:- Write a program to implement Depth First Search Traversal.

Code:-

```
edge(a, b).
edge(b, c).
edge(c, d).
edge(d, a).
edge(b, e).

show_edges :-
  write('Edges of the graph:'), nl,
  edge(X, Y),
  write(X), write(' -> '), write(Y), nl,
  fail.
show_edges.

show_neighbors(Node) :-
  write('Neighbors of '), write(Node), write(':'), nl,
  edge(Node, Neighbor),
  write(Neighbor), nl,
  fail.
show_neighbors(_).

show_graph :-
  show_edges, nl,
  show_neighbors(a),
  show_neighbors(b),
  show_neighbors(c),
  show_neighbors(d).

dfs(Node, Visited) :-
  write('Visiting node: '), write(Node), nl,
  edge(Node, Neighbor),
  not(member(Neighbor, Visited)),
  dfs(Neighbor, [Node | Visited]).
dfs(_, _).

start_dfs(Node) :-
  write('Start DFS from node: '), write(Node), nl,
  dfs(Node, []).
```

Output:-



```
?- start_dfs(a)
|    .
Start DFS from node: a
Visiting node: a
Visiting node: b
Visiting node: c
Visiting node: d
true █
```

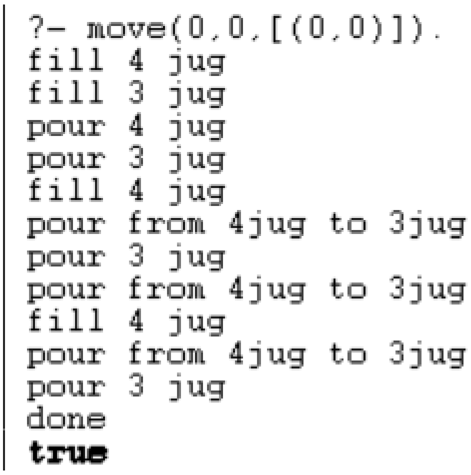
Experiment – 7

Aim:- Write a program to implement the water jug problem using Prolog.

Code:-

```
%input: move(0,0,[(0,0)]).
member(X,[X|_]).
member(X,[Y|Z]):-member(X,Z).

move(X,Y,_):-X=:=2,Y=:=0,write('done'),!.
move(X,Y,Z):-X<4,\+member((4,Y),Z),write("fill 4 jug"),nl,move(4,Y,[(4,Y)|Z]).
move(X,Y,Z):-Y<3,\+member((X,3),Z),write("fill 3 jug"),nl,move(X,3,[(X,3)|z]).
move(X,Y,Z):-X>0,\+member((0,Y),Z),write("pour 4 jug"),nl,move(0,Y,[(0,Y)|Z]).
move(X,Y,Z):-Y>0,\+member((X,0),Z),write("pour 3 jug"),nl,move(X,0,[(X,0)|Z]).
move(X,Y,Z):-P is X+Y,P>=4,Y>0,K is 4-X,M is Y-K,\+member((4,M),Z),write("pour from
3jug to 4jug"),nl,move(4,M,[(4,M)|Z]).
move(X,Y,Z):-P is X+Y,P>=3,X>0,K is 3-Y,M is X-K,\+member((M,3),Z),write("pour from
4jug to 3jug"),nl,move(M,3,[(M,3)|Z]).
move(X,Y,Z):-K is X+Y,K<4,Y>0,\+member((K,0),Z),write("pour from 3jug to
4jug"),nl,move(K,0,[(K,0)|Z]).
move(X,Y,Z):-K is X+Y,K<3,X>0,\+member((0,K),Z),write("pour from 4jug to
3jug"),nl,move(0,K,[(0,K)|Z]).
```

Output:-



```
?- move(0,0,[(0,0)]).
fill 4 jug
fill 3 jug
pour 4 jug
pour 3 jug
fill 4 jug
pour from 4jug to 3jug
pour 3 jug
pour from 4jug to 3jug
fill 4 jug
pour from 4jug to 3jug
pour 3 jug
done
true
```

Experiment – 8

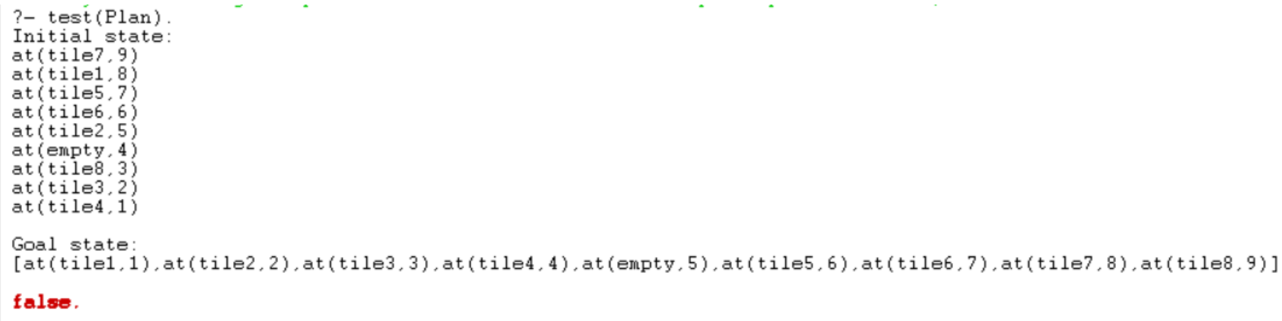
Aim:- Write a program to solve 8 puzzle problem using Prolog.

Code:-

```
% Simple Prolog Planner for the 8 Puzzle Problem

/* This predicate initialises the problem states. The first argument  of solve is
the initial state, the 2nd the goal state, and the third the plan that will be
produced.*/

test(Plan):-
  write('Initial state:'),nl,
  Init= [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4), at(tile2,5),
at(tile6,6), at(tile5,7), at(tile1,8), at(tile7,9)],
  write_sol(Init),
  Goal= [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4), at(empty,5),
at(tile5,6), at(tile6,7), at(tile7,8), at(tile8,9)],
  nl,write('Goal state:'),nl,
  write(Goal),nl,nl,
  solve(Init,Goal,Plan).

solve(State, Goal, Plan):-
  solve(State, Goal, [], Plan).

% Determines whether Current and Destination tiles are a valid move.
is_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1
- Y1).


/* This predicate produces the plan. Once the Goal list is a subset of the current
State the plan is complete and it is written to the screen using write_sol */

solve(State, Goal, Plan, Plan):-
  is_subset(Goal, State), nl,
  write_sol(Plan).

solve(State, Goal, Sofar, Plan):-
  act(Action, Preconditions, Delete, Add),
  is_subset(Preconditions, State),
  \+ member(Action, Sofar),
  delete_list(Delete, State, Remainder),
  append(Add, Remainder, NewState),
  solve(NewState, Goal, [Action|Sofar], Plan).

/* The problem has three operators.
 1st arg = name
 2nd arg = preconditions
 3rd arg = delete list
 4th arg = add list. */
```

```
% Tile can move to new position only if the destination tile is empty & Manhattan
distance = 1
act(move(X,Y,Z),
  [at(X,Y), at(empty,Z), is_movable(Y,Z)],
  [at(X,Y), at(empty,Z)],
  [at(X,Z), at(empty,Y)]).


% Utility predicates.

% Check is first list is a subset of the second

is_subset([H|T], Set):-
  member(H, Set),
  is_subset(T, Set).
is_subset([], _).

% Remove all elements of 1st list from second to create third.

delete_list([H|T], Curstate, Newstate):-
  remove(H, Curstate, Remainder),
  delete_list(T, Remainder, Newstate).
delete_list([], Curstate, Curstate).

remove(X, [X|T], T).
remove(X, [H|T], [H|R]):-
  remove(X, T, R).

write_sol([]).
write_sol([H|T]):-
  write_sol(T),
  write(H), nl.

append([H|T], L1, [H|L2]):-
  append(T, L1, L2).
append([], L, L).

member(X, [X|_]).
member(X, [_|T]):-
  member(X, T).
```

Output:-



```
?- test(Plan).
Initial state:
at(tile7,9)
at(tile1,8)
at(tile5,7)
at(tile6,6)
at(tile2,5)
at(empty,4)
at(tile8,3)
at(tile3,2)
at(tile4,1)

Goal state:
[at(tile1,1),at(tile2,2),at(tile3,3),at(tile4,4),at(empty,5),at(tile5,6),at(tile6,7),at(tile7,8),at(tile8,9)]

false.
```

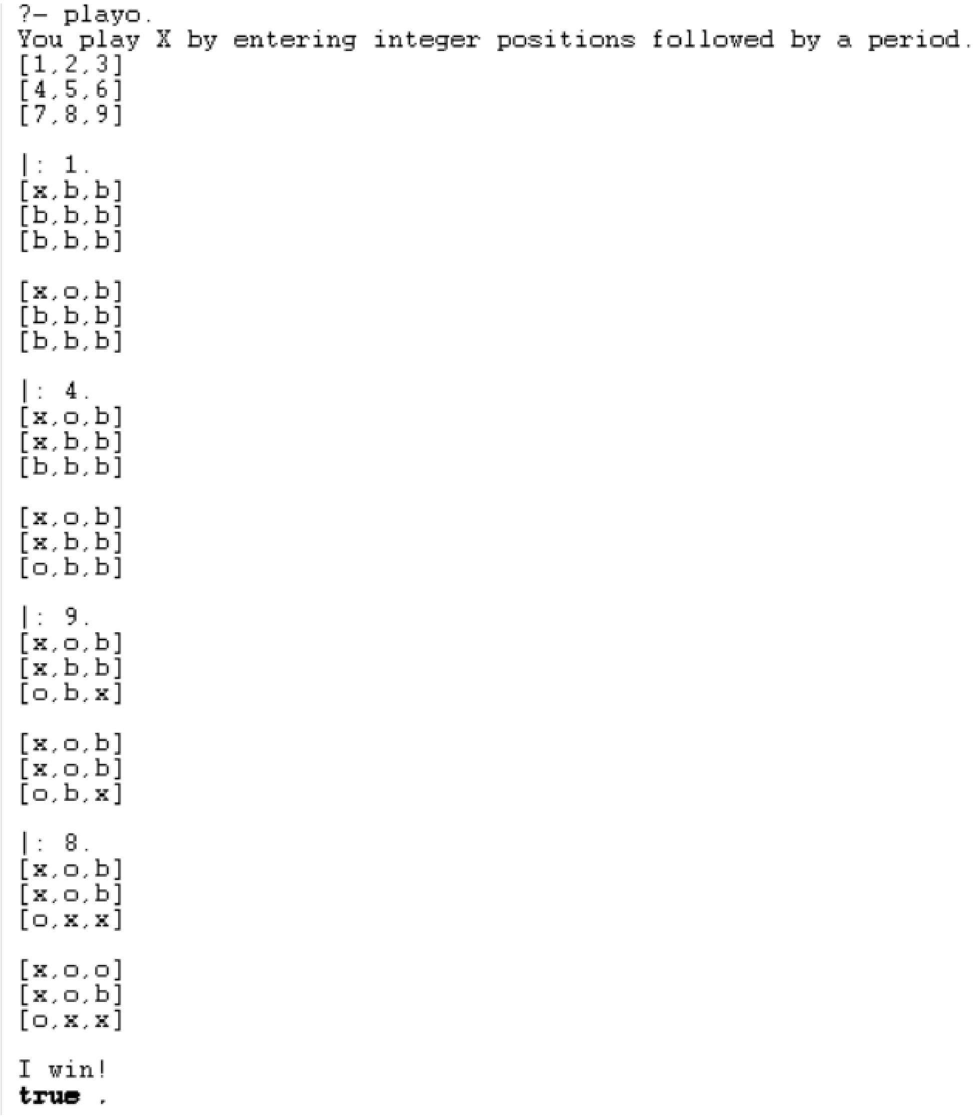
Experiment – 9

Aim:- Write a program to implement a Tic-Tac-Toe game using Prolog.

Code:-

```
win(b, p) :- rowwin(b, p).
win(b, p) :- colwin(b, p).
win(b, p) :- diagwin(b, p).

rowwin(b, p) :- b = [p,p,p,_,_,_,_,_,_].
rowwin(b, p) :- b = [_,_,_,p,p,p,_,_,_].
rowwin(b, p) :- b = [_,_,_,_,_,_,p,p,p].

colwin(b, p) :- b = [p,_,_,p,_,_,p,_,_].
colwin(b, p) :- b = [_,p,_,_,p,_,_,p,_].
colwin(b, p) :- b = [_,_,p,_,_,p,_,_,p].

diagwin(b, p) :- b = [p,_,_,_,p,_,_,_,p].
diagwin(b, p) :- b = [_,_,p,_,p,_,p,_,_].

other(x,o).
other(o,x).

game(b, p) :- win(b, p), !, write([p, p, wins]).
game(b, p) :-
 other(p,Otherp),
 move(b,p,Newb),
 !,
 display(Newb),
 game(Newb,Otherp).

move([b,B,C,D,E,F,G,H,I], p, [p,B,C,D,E,F,G,H,I]).
move([A,b,C,D,E,F,G,H,I], p, [A,p,C,D,E,F,G,H,I]).
move([A,B,b,D,E,F,G,H,I], p, [A,B,p,D,E,F,G,H,I]).
move([A,B,C,b,E,F,G,H,I], p, [A,B,C,p,E,F,G,H,I]).
move([A,B,C,D,b,F,G,H,I], p, [A,B,C,D,p,F,G,H,I]).
move([A,B,C,D,E,b,G,H,I], p, [A,B,C,D,E,p,G,H,I]).
move([A,B,C,D,E,F,b,H,I], p, [A,B,C,D,E,F,p,H,I]).
move([A,B,C,D,E,F,G,b,I], p, [A,B,C,D,E,F,G,p,I]).
move([A,B,C,D,E,F,G,H,b], p, [A,B,C,D,E,F,G,H,p]).

display([A,B,C,D,E,F,G,H,I]) :- write([A,B,C]),nl,write([D,E,F]),nl,
 write([G,H,I]),nl,nl.

selfgame :- game([b,b,b,b,b,b,b,b,b],x).

x_can_win_in_one(b) :- move(b, x, Newb), win(Newb, x).

orespond(b,Newb) :-
 move(b, o, Newb),
 win(Newb, o),
 !.
orespond(b,Newb) :-
 move(b, o, Newb),
 not(x_can_win_in_one(Newb)).
orespond(b,Newb) :-
 move(b, o, Newb).
orespond(b,Newb) :-
 not(member(b,b)),
 !,
 write('Cats game!'), nl,
 Newb = b.

xmove([b,B,C,D,E,F,G,H,I], 1, [x,B,C,D,E,F,G,H,I]).
xmove([A,b,C,D,E,F,G,H,I], 2, [A,x,C,D,E,F,G,H,I]).
xmove([A,B,b,D,E,F,G,H,I], 3, [A,B,x,D,E,F,G,H,I]).
xmove([A,B,C,b,E,F,G,H,I], 4, [A,B,C,x,E,F,G,H,I]).
xmove([A,B,C,D,b,F,G,H,I], 5, [A,B,C,D,x,F,G,H,I]).
xmove([A,B,C,D,E,b,G,H,I], 6, [A,B,C,D,E,x,G,H,I]).
xmove([A,B,C,D,E,F,b,H,I], 7, [A,B,C,D,E,F,x,H,I]).
xmove([A,B,C,D,E,F,G,b,I], 8, [A,B,C,D,E,F,G,x,I]).
xmove([A,B,C,D,E,F,G,H,b], 9, [A,B,C,D,E,F,G,H,x]).
xmove(b, _, b) :- write('Illegal move.'), nl.

playo :- explain, playfrom([b,b,b,b,b,b,b,b,b]).

explain :-
 write('You play X by entering integer positions followed by a period.'),
 nl,
 display([1,2,3,4,5,6,7,8,9]).

playfrom(b) :- win(b, x), write('You win!').
playfrom(b) :- win(b, o), write('I win!').
playfrom(b) :- read(N),
 xmove(b, N, Newb),
 display(Newb),
 orespond(Newb, Newnewb),
 display(Newnewb),
 playfrom(Newnewb).
```

Output:-



```
?- playo.
You play X by entering integer positions followed by a period.
[1,2,3]
[4,5,6]
[7,8,9]

|: 1.
[x,b,b]
[b,b,b]
[b,b,b]

[x,o,b]
[b,b,b]
[b,b,b]

|: 4.
[x,o,b]
[x,b,b]
[b,b,b]

[x,o,b]
[x,b,b]
[o,b,b]

|: 9.
[x,o,b]
[x,b,b]
[o,b,x]

[x,o,b]
[x,o,b]
[o,b,x]

|: 8.
[x,o,b]
[x,o,b]
[o,x,x]

[x,o,o]
[x,o,b]
[o,x,x]

I win!
true .
```

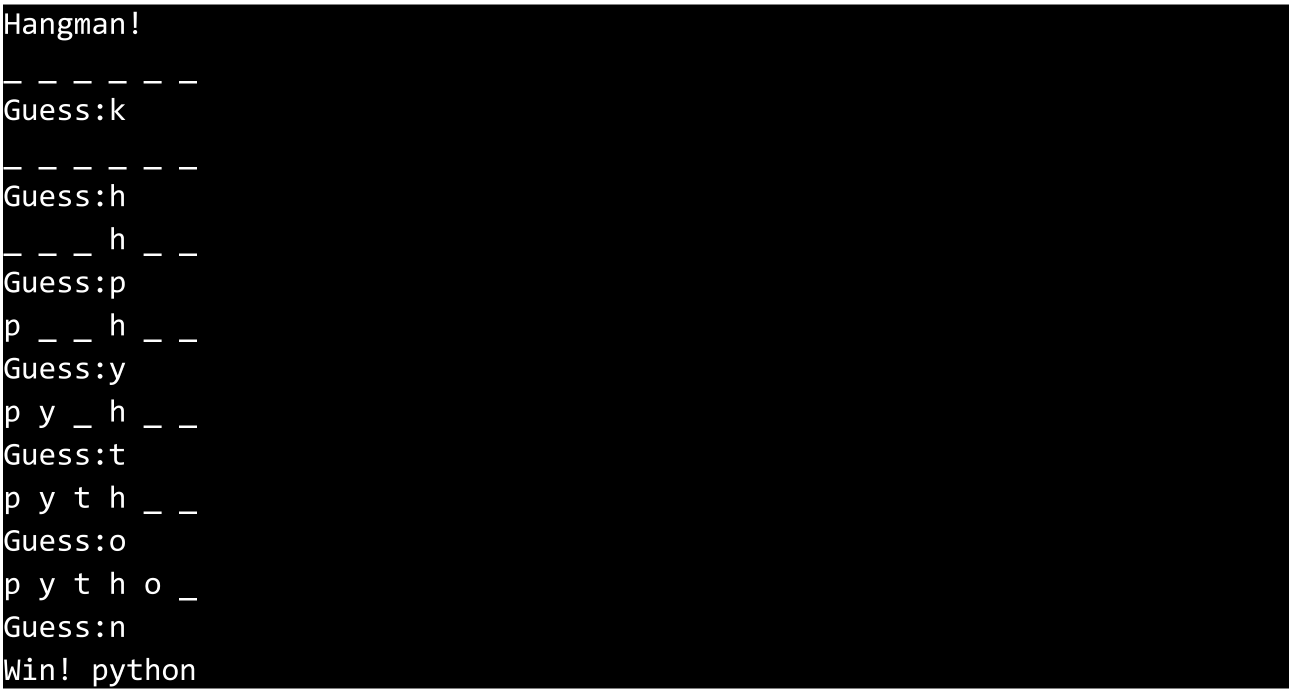
Experiment – 10

Aim:- Write a program to implement the Hangman game using Python.

Code:-

```
import random as r
d=r.choice

def h():
 w=d(["python","java","kotlin","js","hangman","dev","code","YPS"]);g=set();a=6
 print("Hangman!")
 while a:
  print(" ".join(l if l in g else "_" for l in w))
  i=input("Guess:").lower()
  if len(i)!=1 or not i.isalpha()or i in g:continue
  g.add(i);a-=(i not in w)
  if all(l in g for l in w):print("Win!",w);return
 print("Lost!",w)
h()
```

output:



```
Hangman!
_ _ _ _ _ _
Guess:k
_ _ _ _ _ _
Guess:h
_ _ _ h _ _
Guess:p
p _ _ h _ _
Guess:y
p y _ h _ _
Guess:t
p y t h _ _
Guess:o
p y t h o _
Guess:n
Win! python
```

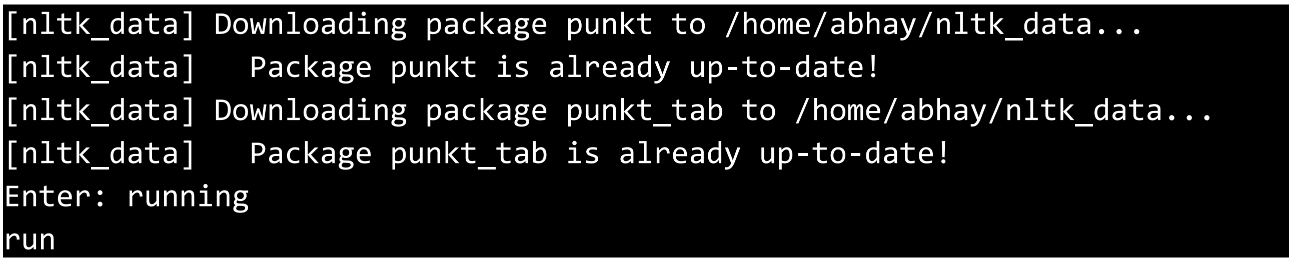
Experiment – 11

Aim:- Write a program to implement stemming for a given sentence using NLTK.

Code:-

```
import nltk
from nltk.stem import PorterStemmer as P
from nltk.tokenize import word_tokenize as w
nltk.download('punkt')
nltk.download('punkt_tab')
s=input("Enter: ")
print(" ".join(P().stem(i) for i in w(s)))
```

output:



```
[nltk_data] Downloading package punkt to /home/abhay/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /home/abhay/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
Enter: running
run
```

Experiment – 12

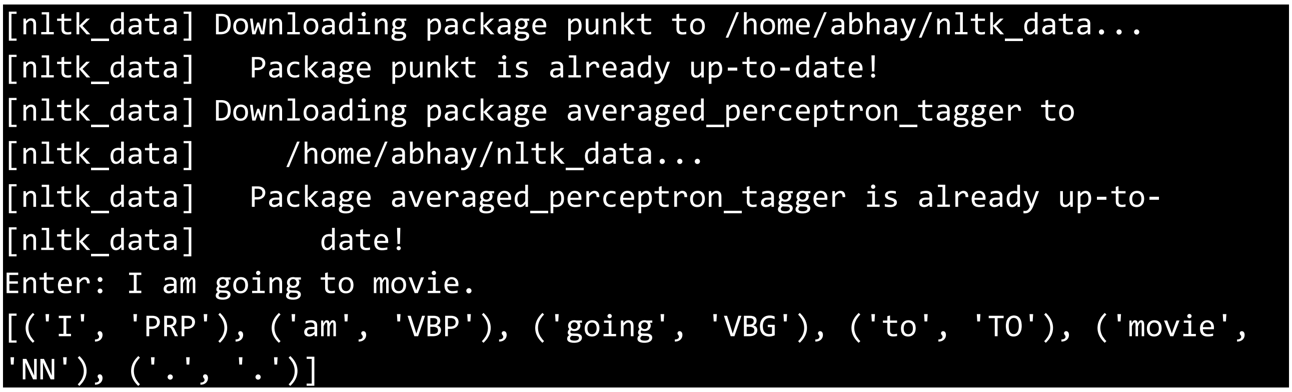
Aim:- Write a program to POS (part of speech) tagging for the given sentence

using NLTK.

Code:-

```
import nltk
from nltk.tokenize import word_tokenize as w
from nltk import pos_tag as p
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
print(p(w(input("Enter: "))))
```

output:



```
[nltk_data] Downloading package punkt to /home/abhay/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/abhay/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
Enter: I am going to movie.
[('I', 'PRP'), ('am', 'VBP'), ('going', 'VBG'), ('to', 'TO'), ('movie',
'NN'), ('.', '.')]
```

Experiment – 13

Aim:- Write a program to implement Lemmatization using NLTK.

Code:-

```
import nltk
from nltk.stem import WordNetLemmatizer as L
from nltk.tokenize import word_tokenize as w
from nltk.corpus import wordnet as wn
nltk.download('punkt')
nltk.download('omw-1.4')
l=L()
def f(wd):return l.lemmatize(wd,pos=wn.VERB)
print(" ".join(f(i) for i in w(input("Enter: "))))
```

Output:

```
[nltk_data] Downloading package punkt to /home/abhay/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /home/abhay/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
Enter: running
run
```

Experiment – 14

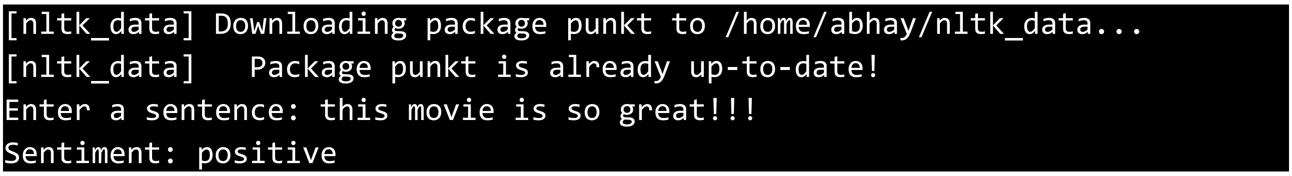
Aim:- Write a program for Text Classification for the given sentence using

NLTK.

Code:-

```
import nltk
from nltk.classify import NaiveBayesClassifier
from nltk.tokenize import word_tokenize
nltk.download('punkt')
def fs(sen):
   return {word: True for word in word_tokenize(sen)}
td = []
with open("td.txt", "r") as f:
   for line in f:
      parts = line.strip().rsplit(" ", 1)  # Split only at the last space
      if len(parts) == 2 and parts[1] in ["positive", "negative"]:
         td.append((parts[0], parts[1]))
train_set = [(fs(text), label) for text, label in td]
classifier = NaiveBayesClassifier.train(train_set)


sen = input("Enter a sen: ")

print("Sentiment:", classifier.classify(fs(sen)))
```

Output:-



```
[nltk_data] Downloading package punkt to /home/abhay/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Enter a sentence: this movie is so great!!!
Sentiment: positive
```