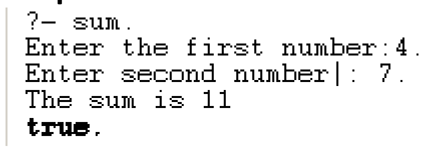
1. WAP in prolog to find the addition of two input numbers asking value from user.

Source Code:

```
sum:-
        write('Enter the first number:'),read(X),
         write('Enter second number'),read(Y),
        S is X+Y,
        write('The sum is '), write(S).
```

Output:



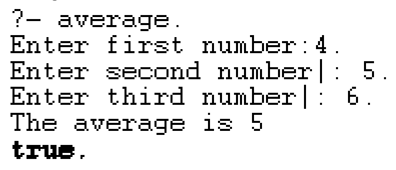
```
?- sum.
Enter the first number:4.
Enter second number|: 7.
The sum is 11
true.
```

2. WAP in prolog to find the average of three input numbers.

Source Code:

```
average:-
        write('Enter first number:'),read(X),
        write('Enter second number'),read(Y),
        write('Enter third number'),read(Z),
        A is (X+Y+Z)/3,
        write('The average is '),write(A).
```

Output:



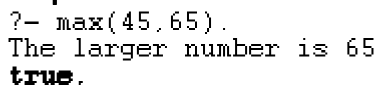
```
?- average.
Enter first number:4.
Enter second number|: 5.
Enter third number|: 6.
The average is 5
true.
```

3. WAP in prolog to compare two input numbers.

Source Code:

```
max(X,Y):-
        X=Y,write('Both number are equal');
        write('The larger number is '),
        X>Y,Z is X,write(Z);
         Z is Y,write(Z).
```

Output:



```
?- max(45,65).
The larger number is 65
true.
```

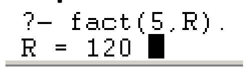
4. WAP in prolog to find the factorial of input number using recursion.

Source Code:

```
fact(0,Result) :-
        Result is 1.
```

```
fact(N,Result) :-
            N > 0, N1 is N-1,
            fact(N1,Result1),
            Result is Result1*N.
```

Output:



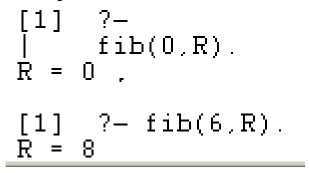
```
?- fact(5,R).
R = 120
```

5. WAP in prolog to find Fibonacci of a given number using recursion.

Source Code:

```
fib(0, 0).
fib(1, 1).
fib(N, R) :-
        N1 is N - 1, N2 is N - 2,
        fib(N1, R1), fib(N2, R2),
        R is R1 + R2.
```

Output:



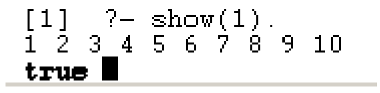
```
[1]  ?-
|    fib(0,R).
R = 0 ,

[1]  ?- fib(6,R).
R = 8
```

6. WAP in prolog to display numbers from 1 to 10.

Source Code:

```
show(10):- write(10).
show(X) :-
        write(X), write(' '),
        Next is X + 1,
        show(Next).
```

Output:



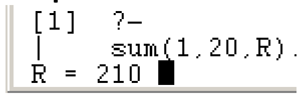
```
[1]  ?- show(1).
1 2 3 4 5 6 7 8 9 10
true
```

7. WAP in prolog to find the sum of numbers from 1 to 20.

Source Code:

```
sum(N, N, N).
sum(M, N, S):-
        N>M,X is M+1,
        sum(X, N, T),
        S is M+T.
```

Output:



```
[1]  ?-
|    sum(1,20,R).
R = 210
```

8. Given the following statement.

    Gold is valuable.
    Sita is female.
    Sita owns gold.
    Ram is the father of Hari.
    Ram gives book to Hari.

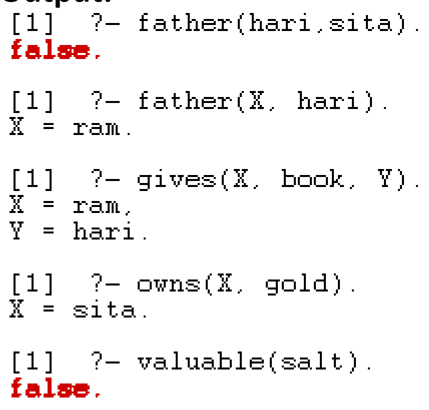
Represent the above statement in the form of facts and answer the following question by running query in prolog.

a.) Is Hari is father of Sita?
b.) Who is father of Hari?
c.) Who gives book to whom?
d.) Who owns gold?
e.) Is salt valuable?

Source Code:

```
valuable(gold).
female(sita).
owns(sita,gold).
father(ram,hari).
gives(ram,book,hari).
```

Output:



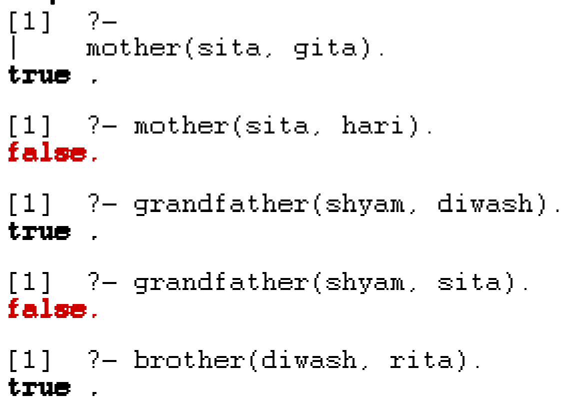
```
[1]  ?- father(hari,sita).
false.

[1]  ?- father(X, hari).
X = ram.

[1]  ?- gives(X, book, Y).
X = ram,
Y = hari.

[1]  ?- owns(X, gold).
X = sita.

[1]  ?- valuable(salt).
false.
```

9. By considering your family tree construct facts about male, female and parent(eg . male(ram), female (sita), parent(ram, hari)).Construct the rule for grandfather, grandmother, father, mother, brother, sister. After that write the query to test different relation.

Source Code:

```
male(shyam).
male(ram).
male(diwash).
female(sita).
female(gita).
female(rita).
parent(shyam, ram).
parent(ram, diwash).
parent(ram, rita).
parent(sita, gita).
parent(gita, rita).
parent(gita, diwash).
parent(sita, ram).
parent(shyam, gita).
mother(M,X):-
        parent(M,X),
        female(M).
father(F,X):-
        parent(F,X),
        male(F).
grandfather(GF,X):-
        parent(GF,F),parent(F,X),
        male(GF).
grandmother(GM, X):-
        parent(GM,M), parent(M,X),
        female(GM).
brother(B,X):-
        male(B),
        parent(P,B), parent(P,X).
sister(S,X):-
        female(S),
        parent(P,S), parent(P,X).
```

Output:



```
[1]  ?-
|    mother(sita, gita).
true .

[1]  ?- mother(sita, hari).
false.

[1]  ?- grandfather(shyam, diwash).
true .

[1]  ?- grandfather(shyam, sita).
false.

[1]  ?- brother(diwash, rita).
true .
```

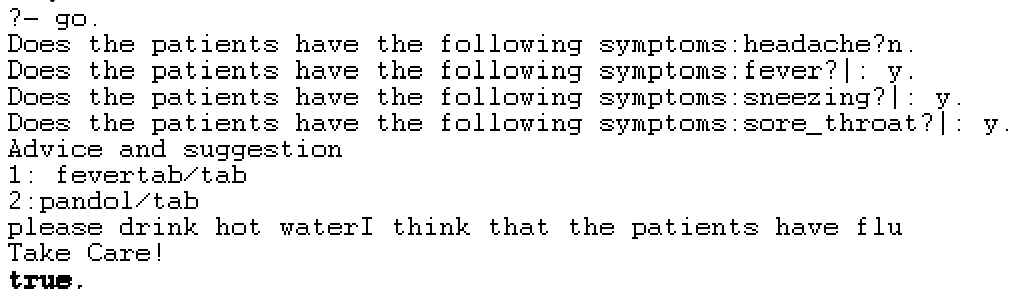
10. Create a medical diagnosis expert system in prolog by gathering data about symptoms of disease and medicine prescribed for that. The expert system can identify which disease has occurred to the person and also suggest some medicine when particular disease is detected.

Source code:

```
go:-
```

```
        hypothesis(Disease),
        write('I think that the patients have '),write(Disease),nl,
        write('Take Care!'),
        undo.
hypothesis(cold):-cold, !.
hypothesis(flu):-flu, !.
cold:-
        verify(headache),
        verify(runny_nose),
        verify(sneezing),
        verify(sore_throat),
        write('Advice and suggesation'),nl,
        write('1: Tylenol/tab'),nl,
        write('2:pandol/tab'),nl,
        write('please wear warm clothes').
flu:-
        verify(fever),
        verify(sneezing),
        verify(sore_throat),
        write('Advice and suggestion'),nl,
        write('1: fevertab/tab'),nl,
        write('2:pandol/tab'),nl,
        write('please drink hot water').
ask(Question):-
        write('Does the patients have the following symptoms:'),
        write(Question),
        write('?'),
        read(Response),
        ((Response == yes; Response ==y)->assert(yes(Question));assert(no(Question)),fail).
:-dynamic yes/1,no/1.
verify(S):-
        (yes(S) -> true;(no(S)-> fail;ask(S))).
undo:-retract(yes(_)),fail.
undo:-retract(no(_)),fail.
undo.
```

Output:



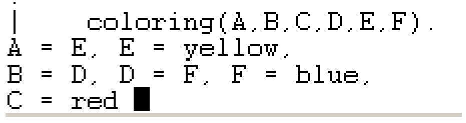
```
?- go.
Does the patients have the following symptoms:headache?n.
Does the patients have the following symptoms:fever?|: y.
Does the patients have the following symptoms:sneezing?|: y.
Does the patients have the following symptoms:sore_throat?|: y.
Advice and suggestion
1: fevertab/tab
2:pandol/tab
please drink hot waterI think that the patients have flu
Take Care!
true.
```

11. WAP in prolog to implement Map coloring problem as a constraint satisfaction problem.

Source Code:

```
coloring(A,B,C,D,E,F):-
        different(A,B),
        different(A,C),
        different(A,D),
        different(A,F),
        different(B,C),
        different(B,E),
        different(C,D),
        different(C,E),
        different(D,E),
        different(E,F).
different(yellow,blue).
different(blue,yellow).
different(yellow,red).
different(red,yellow).
different(blue,red).
different(red,blue).
```

Output:



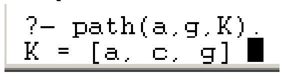
```
|    coloring(A,B,C,D,E,F).
A = E, E = yellow,
B = D, D = F, F = blue,
C = red
```

12. WAP in prolog to implement DFS searching.

Source Code:

```
path(A,G,[A|Z]):-
        childnode(A,G,Z).
childnode(A,G,[G]):-
        child(A,G).
childnode(A,G,[X|L]):-
        child(A,X),
        childnode(X,G,L).
child(a,b).
child(a,c).
child(a,d).
child(b,e).
child(b,f).
child(c,g).
child(g,h).
```

Output:



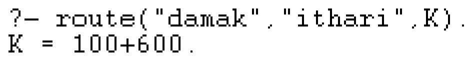
```
?- path(a,g,K).
K = [a, c, g]
```

13. WAP in prolog to implement TSP problem.

Source Code:

```
route(Town1,Town2,Distance):-
        road(Town1,Town2,Distance).
route(Town1,Town2,Distance):-
        road(Town1,X,Dist1),
        route(X,Town2,Dist2),
        Distance=Dist1+Dist2,!.
    road("damak","birtamode",100).
    road("biratnagar","damak",500).
    road("birtamode","biratnagar",300).
    road("birtamode","ithari",600).
    road("biratnagar","ithari",200).
```

Output:



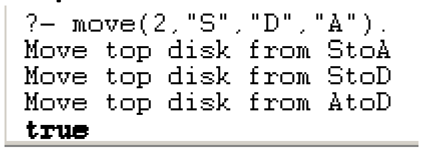
```
?- route("damak","ithari",K).
K = 100+600.
```

14. WAP in prolog to implement Tower of Hanoi problem.

Source Code:

```
move(1,X,Y,_):-
        write('Move top disk from '),
        write(X),
        write('to'),
        write(Y),nl.
move(N,X,Y,Z):-
        N>1,
        M is N-1,
        move(M,X,Z,Y),
        move(1,X,Y,_),
        move(M,Z,Y,X).
```

Output:



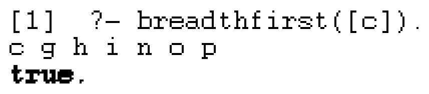
```
?- move(2,"S","D","A").
Move top disk from StoA
Move top disk from StoD
Move top disk from AtoD
true
```

15. WAP in prolog to implement BFS search.

Source Code:

```
:- op(500,xfx,'is_parent').

a is_parent b.    c is_parent g.     f is_parent l.     j is_parent q.
a is_parent c.    c is_parent h.     f is_parent m.     j is_parent r.
a is_parent d.    c is_parent i.     h is_parent n.     j is_parent s.
b is_parent e.    d is_parent j.     i is_parent o.     m is_parent t.
b is_parent f.    e is_parent k.     i is_parent p.

getchildren(Parent, Children) :-
   setof(Child, Parent^is_parent(Parent, Child), Children), !.
getchildren(_, []).
breadthfirst([]) :- !.
breadthfirst([Node|Frontier]) :-
   format('~p ', [Node]),
   getchildren(Node, Children),
   append(Frontier, Children, NewFrontier),
   breadthfirst(NewFrontier).
```

Output:



```
[1]  ?- breadthfirst([c]).
c g h i n o p
true.
```