

1 Write a program to find the family relation and sum of two numbers.

```
male('Ram').
male('Shyam').
male('Arjun').
female('Hari').
female('Subhash').
female('Bijay').
female('Biraj').
female('Kastuv').

father('Ram', 'Arjun').
father('Shyam', 'Bijay').
father('Kastuv', 'Ram').

mother('Hari', 'Arjun').
mother('Subhash', 'Bijay').

husband('Ram', 'Hari').
sister('Bijay', 'Arjun').
brother('Shyam', 'Bijay').

grandfather('Kastuv', 'Arjun').
grandfather('Kastuv', 'Bijay').

grandson('Arjun', 'Kastuv').
granddaughter('Bijay', 'Kastuv').

sum(X, Y, Result) :- Result is X + Y.
```

```
?- [swlab1].
?- father('Ram', 'Arjun').
true.

?- mother('Hari', 'Arjun').
true.

?- husband('Ram', 'Hari').
true.

?- grandson('Arjun', 'Kastuv').
true.

?- granddaughter('Bijay', 'Kastuv').
true.
```

2a Write a program to find the area and perimeter of a rectangle.

```
rectangle_area(Length, Width, Area) :-  
    Area is Length * Width.  
rectangle_perimeter(Length, Width, Perimeter) :-  
    Perimeter is 2 * (Length + Width).  
  
circle_area(Radius, Area) :-  
    Area is 3.14159 * Radius * Radius.  
circle_circumference(Radius, Circumference) :-  
    Circumference is 2 * 3.14159 * Radius.
```

```
?- [swlab2a].  
?- rectangle_area(5, 3, Area).  
Area = 15.  
  
?- rectangle_perimeter(5, 3, Perimeter).  
Perimeter = 16.  
  
?- circle_area(5, Area).  
Area = 78.53975.  
  
?- circle_circumference(5, Circumference).  
Circumference = 31.4159.
```

2b.

Write a program to find the sum, difference, multiplication, division, and integer division.

```
sum(X, Y, Result) :- Result is X + Y.  
difference(X, Y, Result) :- Result is X - Y.  
multiplication(X, Y, Result) :- Result is X * Y.  
division(X, Y, Result) :- Y \= 0, Result is X / Y.  
integer_division(X, Y, Result) :- Y \= 0, Result is X // Y.
```

```
square(X, Result) :- Result is X * X.  
cube(X, Result) :- Result is X * X * X.  
modulus(X, Y, Result) :- Y \= 0, Result is X mod Y.
```

```
?- [swlab2b].  
?- sum(5, 3, Result).  
Result = 8.  
  
?- difference(5, 3, Result).  
Result = 2.  
  
?- multiplication(5, 3, Result).  
Result = 15.  
  
?- division(6, 3, Result).  
Result = 2.  
  
?- integer_division(7, 3, Result).  
Result = 2.  
  
?- square(4, Result).  
Result = 16.  
  
?- cube(3, Result).  
Result = 27.  
  
?- modulus(7, 3, Result).  
Result = 1.
```

3a Write a program to find factorial.

```
% Factorial predicate
factorial(0, 1). % Base case: factorial of 0 is 1
factorial(N, Result) :-
    N > 0,          % Ensure N is positive
    N1 is N - 1,    % Decrement N
    factorial(N1, TempResult), % Recursive call
    Result is N * TempResult. % Calculate factorial
```

```
?- [swlab3a].
?- factorial(5, Result).
Result = 120.
```

3b Write a program to find fibonacci.

```
% Fibonacci predicate
fibonacci(0, 0). % Base case: Fibonacci(0) is 0
fibonacci(1, 1). % Base case: Fibonacci(1) is 1
fibonacci(N, Result) :-
    N > 1,          % Ensure N is greater than 1
    N1 is N - 1,    % Decrement N
    N2 is N - 2,    % Decrement N by 2
    fibonacci(N1, Temp1), % Recursive call for N-1
    fibonacci(N2, Temp2), % Recursive call for N-2
    Result is Temp1 + Temp2. % Calculate Fibonacci
```

```
?- [swlab3b].
?- fibonacci(6, Result).
Result = 8.
```

4a Write a program to find the GCD and LCM of a given number.

```
gcd(X, 0, X) :- X > 0.  
gcd(X, Y, Result) :-  
    Y > 0,  
    R is X mod Y,  
    gcd(Y, R, Result).  
  
lcm(X, Y, Result) :-  
    gcd(X, Y, Gcd),  
    Result is (X * Y) // Gcd.
```

```
?- [swlab4a].  
?- gcd(48, 18, Gcd).  
Gcd = 6.  
  
?- lcm(4, 5, Lcm).  
Lcm = 20.
```

4b Write a program to find the relation.

```
male('Ram').  
male('Shyam').  
male('Arjun').  
male('Bijay').  
female('Hari').  
female('Subhash').  
female('Kastuv').  
  
parent('Ram', 'Arjun').  
parent('Shyam', 'Bijay').  
parent('Hari', 'Arjun').  
parent('Subhash', 'Bijay').  
  
sibling(X, Y) :- parent(P, X), parent(P, Y), X \= Y.  
grandparent(GP, GC) :- parent(GP, P), parent(P, GC).  
cousin(X, Y) :- parent(P1, X), parent(P2, Y), sibling(P1, P2).
```

```
?- [swlab4b].  
?- sibling('Arjun', 'Bijay').  
true.  
  
?- grandparent('Kastuv', 'Arjun').  
true.  
  
?- cousin('Arjun', 'Bijay').  
false.
```

4c Write a program to find the solution to a tower of Hanoi problem.

```
tower_of_hanoi(1, Source, Target, _) :-  
    write('Move disk from '), write(Source), write(' to '), write(Target),  
    nl.  
  
tower_of_hanoi(N, Source, Target, Auxiliary) :-  
    N > 1,  
    M is N - 1,  
    tower_of_hanoi(M, Source, Auxiliary, Target),  
    write('Move disk from '), write(Source), write(' to '), write(Target),  
    nl,  
    tower_of_hanoi(M, Auxiliary, Target, Source).
```

```
?- [swlab4c].  
?- tower_of_hanoi(3, 'A', 'C', 'B').  
Move disk from A to C  
Move disk from A to B  
Move disk from C to B  
Move disk from A to C  
Move disk from B to A  
Move disk from B to C  
Move disk from A to C  
true
```

5a Write a program to find an area of a triangle.

area_of_triangle(Base, Height, Area) :-
Area is 0.5 * Base * Height.

```
?- [swlab5a].  
?- area_of_triangle(5, 10, Area).  
Area = 25.0.
```

5b Write a program to find the cube of a given number.

cube(X, Result) :-
Result is X * X * X.

```
?- [swlab5b].  
?- cube(3, Result).  
Result = 27.
```

6a Write a program to concatenate.

```
concatenate([], List, List).
concatenate([Head|Tail], List, [Head|Result]) :-
    concatenate(Tail, List, Result).
```

```
?- [swlab6a].
?- concatenate([1, 2], [3, 4], Result).
Result = [1, 2, 3, 4].
```

6b Write a program to delete.

```
delete_element(_, [], []).
delete_element(Element, [Element|Tail], Tail).
delete_element(Element, [Head|Tail], [Head|Result]) :-
    delete_element(Element, Tail, Result).
```

```
?- [swlab6b].
?- delete_element(3, [1, 2, 3, 4], Result).
Result = [1, 2, 4].
```

6c Write a program to count the length of a list.

```
length_of_list([], 0).
length_of_list([_|Tail], Length) :-
    length_of_list(Tail, TailLength),
    Length is TailLength + 1.
```

```
?- [swlab6c].
?- length_of_list([1, 2, 3], Length).
Length = 3.
```

6d Write a program to check whether the given member is in the list or not.

```
reverse_list([], []).
reverse_list([Head|Tail], Reversed) :-
    reverse_list(Tail, ReversedTail),
    append(ReversedTail, [Head], Reversed).
```

```
?- [swlab6d].
?- reverse_list([1, 2, 3], Reversed).
Reversed = [3, 2, 1].
```


7a Write a program to find the greatest number among given numbers.

```
greatest([X], X).  
greatest([Head|Tail], Greatest) :-  
    greatest(Tail, TempGreatest),  
    Greatest is max(Head, TempGreatest).
```

```
?- [swlab7a].  
?- greatest([3, 1, 4, 1, 5, 9], G).  
G = 9.
```

7b Write a program to check if the number is a palindrome.

```
palindrome(List) :-  
    reverse_list(List, List).
```

```
?- [swlab7b].  
?- palindrome([1, 2, 3, 2, 1]).  
true.
```

8a Write a program to find the average of a given number.

```
average(List, Average) :-  
    length_of_list(List, Length),  
    sum_list(List, Sum),  
    Average is Sum / Length.
```

```
sum_list([], 0).  
sum_list([Head|Tail], Sum) :-  
    sum_list(Tail, TailSum),  
    Sum is Head + TailSum.
```

```
?- [swlab8a].  
?- average([1, 2, 3, 4], Avg).  
Avg = 2.5.
```

8b Write a program to find the factor of given number.

```
factors(0, []).  
factors(N, Factors) :-  
    findall(X, (between(1, N, X), N mod X == 0), Factors).
```

```
?- [swlab8b].  
?- factors(10, Factors).  
Factors = [1, 2, 5, 10].
```

9a Write a program to find the relation. (Who is mortal?)

```
mortal(X) :- human(X).  
human(socrates).
```

```
?- [swlab9a].  
?- mortal(X).  
X = socrates.
```

9b . Write a program to find the relation. (Which locations are in Asia ?)

```
location('Tokyo', 'Asia').  
location('Beijing', 'Asia').  
location('New York', 'North America').  
  
in_asia(X) :- location(X, 'Asia').
```

```
?- [swlab9b].  
?- in_asia(X).  
X = 'Tokyo' ;  
X = 'Beijing' ;  
false.
```

10a Write a program to find the maximum number of given numbers.

```
max(X, Y, X) :- X >= Y.  
max(X, Y, Y) :- Y > X.
```

```
max_list([X], X).  
max_list([X|Rest], M) :-  
    max_list(Rest, R),  
    max(X, R, M).
```

```
?- [swlab10a].  
?- max_list([3, 5, 2, 9, 1], M).  
M = 9.
```

10b Write a program to find the greatest number among given number.

```
greatest([X], X).  
greatest([X|Rest], G) :-  
    greatest(Rest, R),  
    (X > R -> G = X ; G = R).
```

```
?- [swlab10b].  
?- greatest([3, 5, 2, 9, 1], G).  
G = 9.
```

11a

What are the values of S, M, E, N, D, O, R, and Y that satisfy the equation
SEND + MORE = MONEY?

```
solve(S, E, N, D, O, R, Y) :-  
    Digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],  
    select(S, Digits, Rest),  
    S \= 0,  
    select(E, Rest, Rest1),  
    select(N, Rest1, Rest2),  
    select(D, Rest2, Rest3),  
    select(M, Rest3, Rest4),  
    M \= 0,  
    select(O, Rest4, Rest5),  
    select(R, Rest5, Rest6),  
    select(Y, Rest6, _),  
    SEND is 1000*S + 100*E + 10*N + D,  
    MORE is 1000*M + 100*O + 10*R + E,  
    MONEY is 10000*M + 1000*O + 100*N + 10*E + Y,  
    SEND + MORE == MONEY.
```

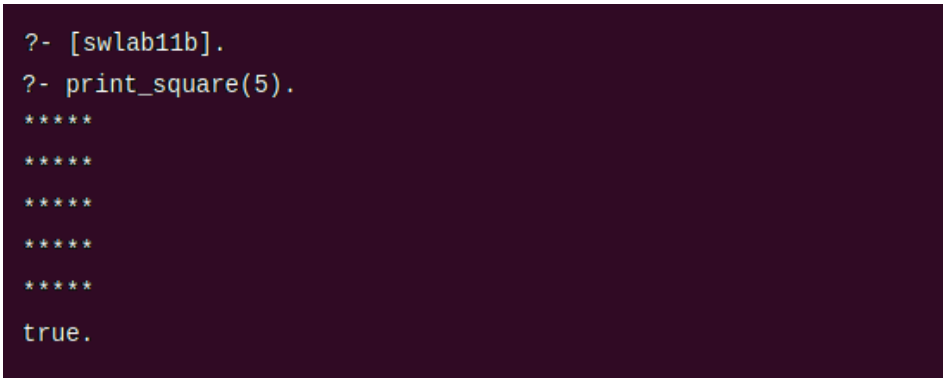
?- solve(S, E, N, D, O, R, Y).

```
?- [swlab11a].  
?- solve(S, E, N, D, O, R, Y).  
S = 9,  
E = 5,  
N = 6,  
D = 7,  
O = 0,  
R = 8,  
Y = 2.
```

11b Write a program to print a square of stars with a side length X.

```
print_square(0).
print_square(X) :-
    X > 0,
    print_row(X),
    NewX is X - 1,
    print_square(NewX).
```

```
print_row(0).
print_row(X) :-
    X > 0,
    write('*'),
    NewX is X - 1,
    print_row(NewX).
```



```
?- [swlab11b].
?- print_square(5).
*****
*****
*****
*****
*****
true.
```

12a

Write a program to find the path from the start node to the goal node using breadth-first search.

```
:- dynamic edge/2.
```

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

```
bfs(Start, Goal, Path) :-  
    bfs([[Start]], Goal, RevPath),  
    reverse(RevPath, Path).
```

```
bfs([[Goal|Visited]|_], Goal, [Goal|Visited]).  
bfs([[Current|Visited]|Rest], Goal, Path) :-  
    findall(Next, (edge(Current, Next), \+ member(Next, [Current|Visited])),  
    add_paths(NextNodes, [Current|Visited], NewPaths),  
    append(Rest, NewPaths, UpdatedPaths),  
    bfs(UpdatedPaths, Goal, Path).
```

```
add_paths([], _, []).
```

```
add_paths([H|T], Visited, [[H|Visited]|NewPaths]) :-  
    add_paths(T, Visited, NewPaths).
```

```
?- [swlab12a].  
?- bfs(a, f, Path).  
Path = [a, b, d, f] ;  
Path = [a, c, d, f] ;  
Path = [a, b, e, f] ;  
Path = [a, c, d, f] ;  
false.
```

12b

Write a program to find the path from the start node to the goal node using depth-first search.

```
:- dynamic edge/2.
```

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

```
dfs(Start, Goal, Path) :-  
    dfs(Start, Goal, [Start], Path).
```

```
dfs(Goal, Goal, Visited, Path) :-  
    reverse(Visited, Path).
```

```
dfs(Current, Goal, Visited, Path) :-  
    edge(Current, Next),  
    \+ member(Next, Visited),  
    dfs(Next, Goal, [Next|Visited], Path).
```

```
?- [swlab12b].  
?- dfs(a, f, Path).  
Path = [a, b, e, f] ;  
Path = [a, b, d, f] ;  
Path = [a, c, d, f] ;  
Path = [a, c, f] ;  
false.
```