**1. Write a Prolog program to calculate the factorial of a given number.**

```
factorial(0, 1).
factorial(N, F) :-
  N > 0,
  N1 is N - 1,
  factorial(N1, F1),
  F is N * F1.
```

## OUTPUT

### Example - 1

```
?- factorial(9, Factorial).
Factorial = 362880 .
```

### Example - 2

```
?- factorial(14, Factorial).
Factorial = 87178291200 .
```

**2. Write a Prolog program to calculate the nth Fibonacci number.**

```
fibonacci(0, 0).
fibonacci(1, 1).
fibonacci(N, F) :-
  N > 1,
  N1 is N - 1,
  N2 is N - 2,
  fibonacci(N1, F1),
  fibonacci(N2, F2),
  F is F1 + F2.
```

## OUTPUT

### Example - 1

```
?- fibonacci(9, Fibonacci).
Fibonacci = 34 .
```

### Example - 2

```
?- fibonacci(17, Fibonacci).
Fibonacci = 1597 .
```

**3. Write a Prolog program to find the maximum of two numbers.**

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

## OUTPUT

### Example - 1

```
?- max(12, 14, Max).
Max = 14.
```

### Example - 2

```
?- max(-73, 14, Max).
Max = 14.
```

**4. Write a Prolog program to implement append for two lists.**

```
append_list([], L, L).
append_list([H|T], L, [H|R]) :-
  append_list(T, L, R).
```

## OUTPUT

### Example - 1

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

### Example - 2

```
?- append_list([1, 2, 3, 6, 11, 43], [7, 1, 90, 22, 3, 4], Result).
Result = [1, 2, 3, 6, 11, 43, 7, 1, 90|...].
```

**5. Write a Prolog program to implement reverse(List, ReversedList) that reverses lists.**

```
% Base case: reversing an empty list gives an empty list
reverse([], []).

% Recursive case
reverse([H|T], R) :-
  reverse(T, RT),
  append(RT, [H], R).
```

## OUTPUT

### Example - 1

```
?- reverse([1, 2, 3, 7, 2, 9], Reverse).
Reverse = [9, 2, 7, 3, 2, 1].
```

### Example - 2

```
?- reverse([1, 2, 3, 7, 2, 33, 129, 43, 9], Reverse).
Reverse = [9, 43, 129, 33, 2, 7, 3, 2, 1].
```

**6. Write a Prolog program to implement palindrome(List).**

```
palindrome(L) :-
  reverse(L, L).

reverse([], []).
reverse([H|T], R) :-
  reverse(T, RevT),
  append(RevT, [H], R).
```

## OUTPUT

### Example - 1

```
?- palindrome([1, 3, 5, 3, 1]).
true.
```

### Example - 2

```
?- palindrome([1, 3, 5, 3, 1, 2]).
false.
```

**7. Write a Prolog program to implement maxlist(List, Max) so that Max is the greatest number in the list of numbers List.**

```
% Base case: max of single-element list is that element
maxlist([X], X).

% Recursive case: compare head with max of tail
maxlist([H|T], Max) :-
  maxlist(T, MaxTail),
  (H >= MaxTail -> Max = H ; Max = MaxTail).
```

## OUTPUT

### Example - 1

```
?- maxlist([1, 3, 5, -32, 22, 42, 100, -120, 101], Max).
Max = 101 .
```

### Example - 2

```
?- maxlist([-1, 0, 1, 2, 3, -4, 5, -66, -3], Max).
Max = 5
```

**8. Write a Prolog program to implement sumlist(List, Sum) so that Sum is the sum of a given list of numbers List.**

```
% Base case: sum of empty list is 0
sumlist([], 0).

% Recursive case
sumlist([H|T], Sum) :-
  sumlist(T, Rest),
  Sum is H + Rest.
```

## OUTPUT

### Example - 1

```
?- sumlist([1, 2, 4, 3, 5, 10, 0, 9, 7, 8, 6], SumOf0to10).
SumOf0to10 = 55.
```

### Example - 2

```
?- sumlist([1, 3, 5, 7, 9, 11, 13, 15], SumOfOdd).
   SumOfOdd = 64.
```

**9. Write a Prolog program to implement evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively.**

```
% Base case: empty list has even length
evenlength([]).
evenlength([_|T]) :- oddlength(T).

% One-element list has odd length
oddlength([_]).
oddlength([_|T]) :- evenlength(T).
```

## OUTPUT

### Example - 1

```
?- evenlength([1, 2, 3, 4, 5, 7, 9]).
false.
```

### Example - 2

```
?- evenlength([1, 2, 3, 4, 5, 7, 9, 10]).
true .
```

### Example - 3

```
?- oddlength([1, 2, 3, 4, 5, 7, 9, 23, 24]).
true .
```

### Example - 4

```
?- oddlength([1, 2, 3, 4, 5, 7, 9, 23, 24, 19, 20, 21]).
false.
```