Exercises

Describe the result

```
p([1,2,3,4,5,6,7,8,9]).
:-p(X).
:- p([X|Y]).
:- p([X,Y]).
:- p([X,Y|Z]).
:- p([_|X]).
```

Check if one number is in the list

is_list(T) = true se T is a list

false if T is not a list

is_list([]).

is_list([_|L]) :- is_list(L).

Example:

:- is_list([1,2,3]).

true

:- is_list([a|b]).

false

Check if a term belongs to a list

member(T,L) "T is one element of list L”

member(T, [T | _]).

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

:- member(2, [1,2,3]).

true

:- member(1, [2,3]).

false

:- member(X, [1,2,3]).

X=1;

X=2;

X=3;
false

Define the Prolog predicate:

no_common_elements(List1, List2) that is true if both List1 and List2 have no elements in common.

Method 1:

```
no_common_elements([], _).
no_common_elements([H1|L1], L2) :- \+ member(H1,L2), no_common_elements(L1, L2).
```

Method 2

```
no_common_elements(List1, List2) :- member(Element, List1), memberchk(Element, List2), !, fail.

no_common_elements (_,_).
```

Method 3

```
no_common_elements (List1, List2) :- \+(member(X, List1), member(X, List2)).
```

Define the length of a list.

length([],0).

length([_|L],N) :- length(L,N1), N is N1 + 1.

Example:

?-length([a,b,d,g],L).

L=4 ;

False.

?-length(X,Y).

X=[] Y=0 ;

X=[_1] Y=1 ; infinit

Write a predicate which, given a term T and a list L, counts the occurrences of T in L.

conta(T,L,N) "N is the number of occurrences of the term T in list L"

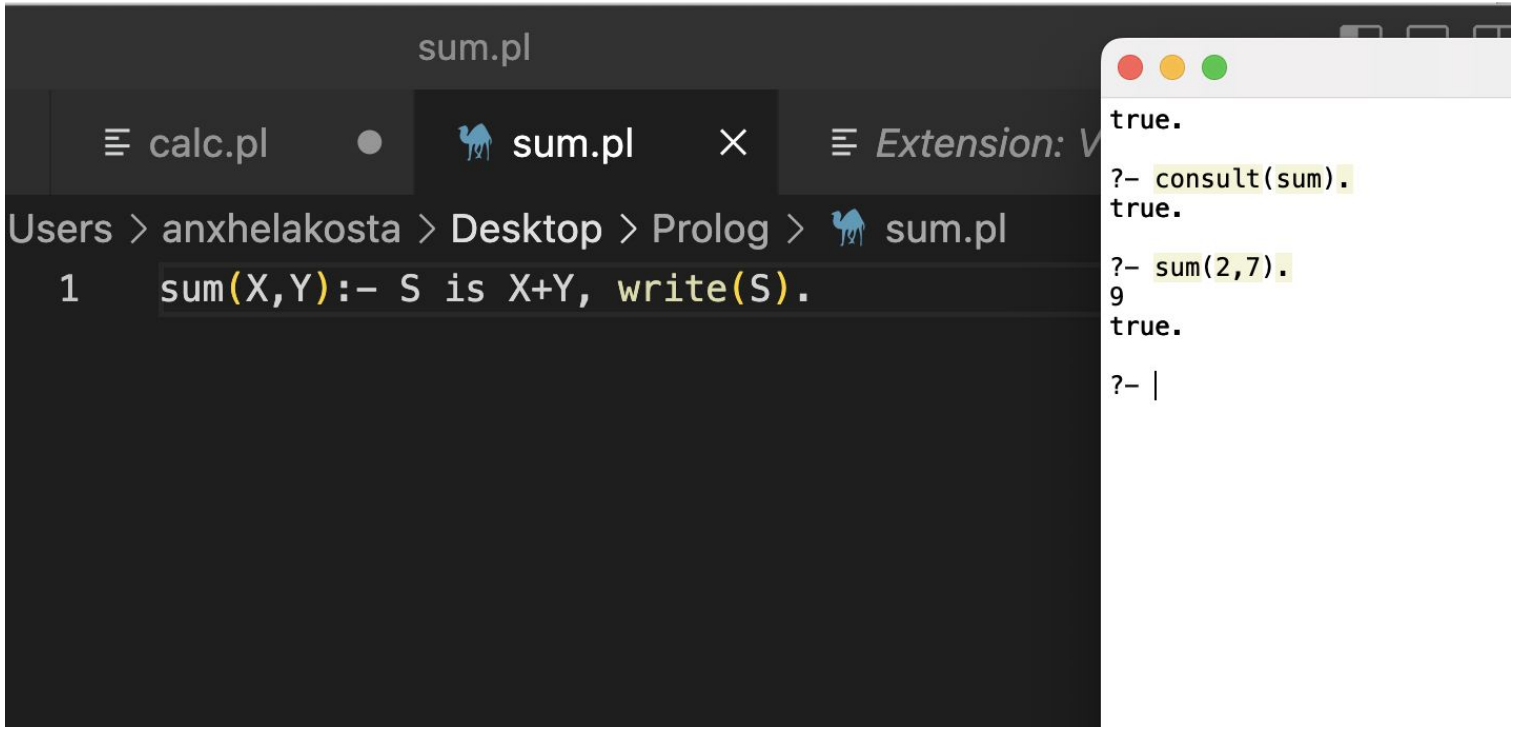
Example:

?- conta(a,[b,a,a,b,c,a],N).

N = 3;

false

Sum of two numbers



Max of two numbers

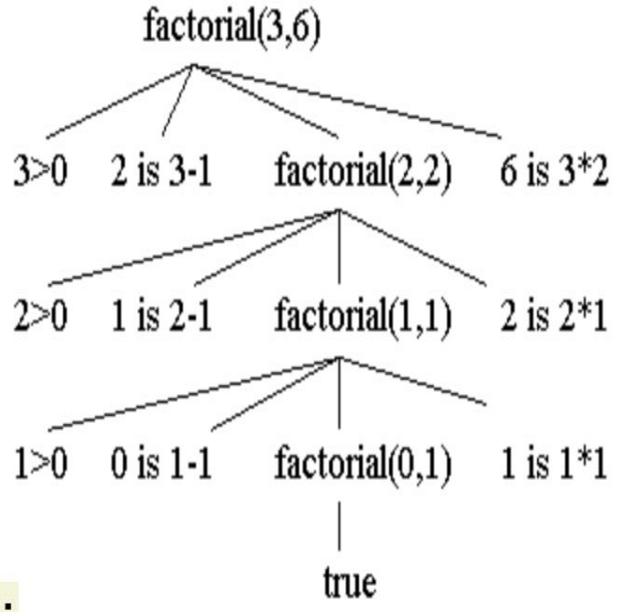
```
max(X,Y):-
    X=Y,write("both are eqaul");
    X>Y, M is X,
    write(M);
    M is Y, write(Y).
```

```
?- max(-100,-1001)
-100
true
```

Find factorial of a nr

```
/* prolog tutorial  2.2 Two Factorial Definitions */

factorial(0,1).

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

/*-----------------------------*/

factorial(0,F,F).

factorial(N,A,F) :-
    N > 0,
    A1 is N*A,
    N1 is N -1,
    factorial(N1,A1,F).
```

```
?- factorial(4,X).
X = 24
```



- **Write a program in PROLOG to implement generate_fib(N,T) where T represents the Nth term of the fibonacci series.**

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for $n > 1$.

```
fib(1,0).
fib(2,1).
fib(N,X):- N1 is N-1,N2 is N-2,fib(N1,X1),fib(N2,X2),X is X1+X2,!.
```

```
?- fib(3,X).
X = 1.

?- fib(10,X).
X = 34.
```

Write a Prolog program to implement GCD of two numbers.

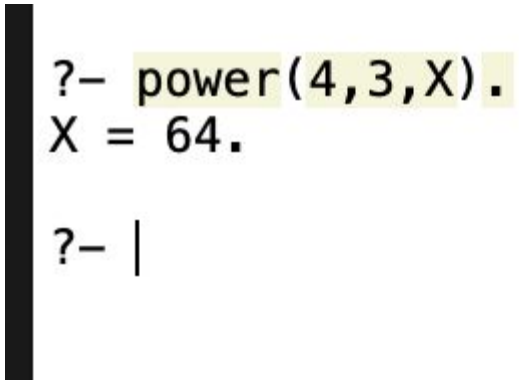
GCD-> greatest common factor of 15 and 10 is 5, since both the numbers can be divided by 5.

```
gcd(0,A,A):-!.
gcd(A,0,A):-!.
 gcd(A,B,R):-B1 is mod(A,B),gcd(B,B1,R).

/
```

Homework: Write A Prolog Program To Compute LCM (Least Common Multiple)

Write a Prolog program to implement power (Num,Pow, Ans) : where Num is raised to the power Pow to get Ans

```
power(X,0):- !.
power(Num,Pow,Ans):-Ans is Num^Pow.
```



Prolog program to implement multi (N1, N2, R) : where N1 and N2 denotes the numbers to be multiplied and R represents the result

```
multi(X,0).

multi(N1,N2,R):- R is N1*N2.
```

Homework: Consider a cyclic directed graph [edge (p, q), edge (q, r), edge (q, r), edge (q, s), edge (s,t)] where edge (A,B) is a predicate indicating directed edge in a graph from a node A to a node B. Write a program to check whether there is a route from one node to another node.

Write a Prolog program to implement memb(X, L): to check whether X is a member of L or not

```
memb(X,[X|Tail]).

memb(X,[Head|Tail]):-memb(X,Tail)
```

Write a Prolog program to implement conc (L1, L2, L3) where L2 is the list to be appended with L1 to get the resulted list L3.

```
conc([],L,L).

conc([X|L1],L2,[X|L3]):-conc(L1,L2,L3)
```

Write a Prolog program to implement reverse (L, R) where List L is original and List R is reversed list.

```
?- reverse([1,2,3],R).
R = [3, 2, 1].

?-
```

```
append([],L,L).
append([X|L1],L2,[X|L3]):- append(L1,L2,L3).

reverse([],[]).

reverse([H|T],R):- reverse(T,L1),append(L1,[H],R).
```

```
?- append([1,2,3],[3,4,5],X).
X = [1, 2, 3, 3, 4, 5].

?-
```

Write a program in PROLOG to implement palindrome (L) which checks whether a list L is a palindrome or not.

a word, verse, or sentence that reads the same backward or forward

```
app([],L,L).

app([X|L1],L2,[X|L3]):- app(L1,L2,L3).

 pal([]).

pal([_]).

pal(Plist):-app([H|T],[H],Plist),pal(T).
```

Find the minimum of a list

```
minElem([Min], Min).

minElem([Min|Tail], Min) :- minElem(Tail, TailMin), Min =< TailMin.

minElem([Head|Tail], TailMin) :- minElem(Tail, TailMin), Head > TailMin.
```