Final Practical File

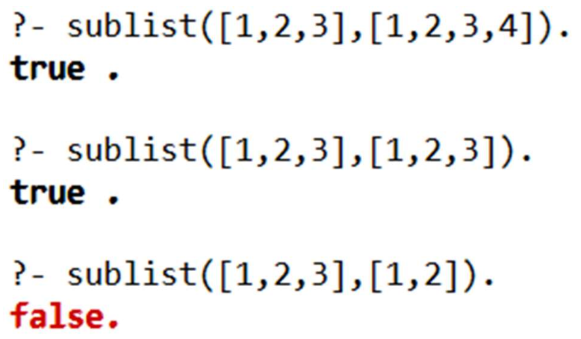
Name: Ravish Ranjan
Course: BSc. Hons. Computer Science
Semester: 6th Semester
Paper: Artificial Intelligence
Roll no.: 21058570040

Que 1 WAP to check is a list is a subset of another.

Ans

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

#Output



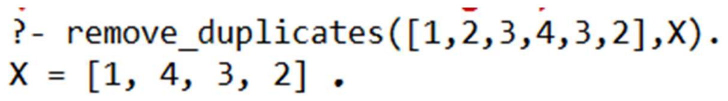
```
?- sublist([1,2,3],[1,2,3,4]).
true .

?- sublist([1,2,3],[1,2,3]).
true .

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

Que 2 WAP to remove duplicates from a list.

Ans

```
remove_duplicates([],[]).
remove_duplicates([H | T], List) :-
     member(H, T),
     remove_duplicates( T, List).

remove_duplicates([H | T], [H|T1]) :-
      not(member(H, T)),
      remove_duplicates( T, T1).
```

#Output



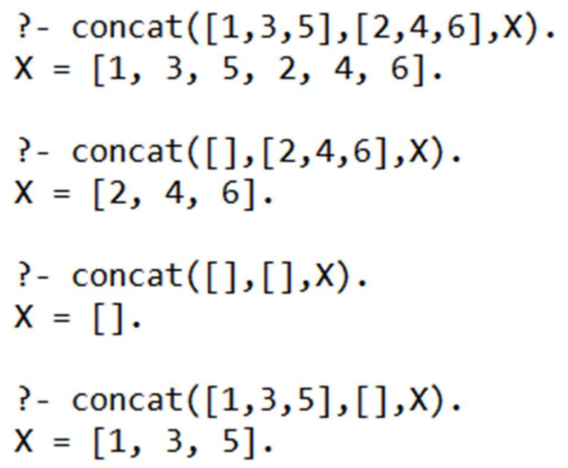
```
?- remove_duplicates([1,2,3,4,3,2],X).
X = [1, 4, 3, 2] .
```

Que 3 WAP to concatenate two lists.

Ans

```
concat([],L,L).
concat([H|T],L,[H|TR]):- concat(T,L,TR).
```

#Output



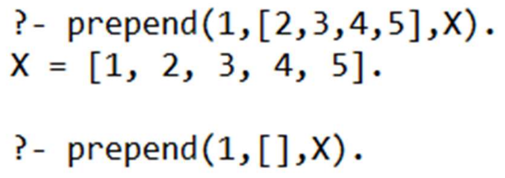
```
?- concat([1,3,5],[2,4,6],X).
X = [1, 3, 5, 2, 4, 6].

?- concat([],[2,4,6],X).
X = [2, 4, 6].

?- concat([],[],X).
X = [].

?- concat([1,3,5],[],X).
X = [1, 3, 5].
```

Que 4 WAP to prepend an element in a list.

Ans

```
prepend(I,[],[I]).
prepend(I,L,R):- concat([I],L,R).
```

#Output



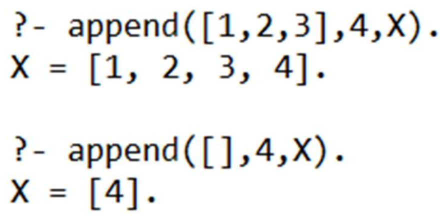
```
?- prepend(1,[2,3,4,5],X).
X = [1, 2, 3, 4, 5].

?- prepend(1,[],X).
```

Que 5 WAP to append an element in list.

Ans

```
append(L,I,R):- concat(L,[I],R).
```

#Output



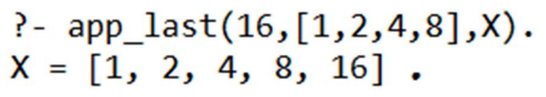
```
?- append([1,2,3],4,X).
X = [1, 2, 3, 4].

?- append([],4,X).
X = [4].
```

Que 6 WAP to append an element in list using recursion.

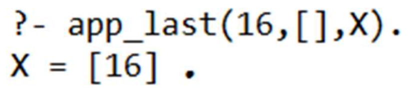
Ans

```
app_last(I,[],[I]).
app_last(I,[H|T],[H|TR]):- app_last(I,T,TR).
```

#Output



```
?- app_last(16,[1,2,4,8],X).
X = [1, 2, 4, 8, 16] .
```



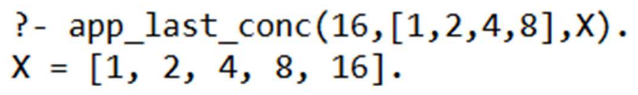
```
?- app_last(16,[],X).
X = [16] .
```

Que 7 WAP to append an element in list using concatenation.

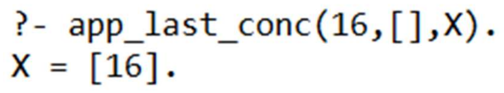
Ans

```
app_last_conc(I,L,R):- concat(L,[I],R).
```

#Output



```
?- app_last_conc(16,[1,2,4,8],X).
X = [1, 2, 4, 8, 16].
```



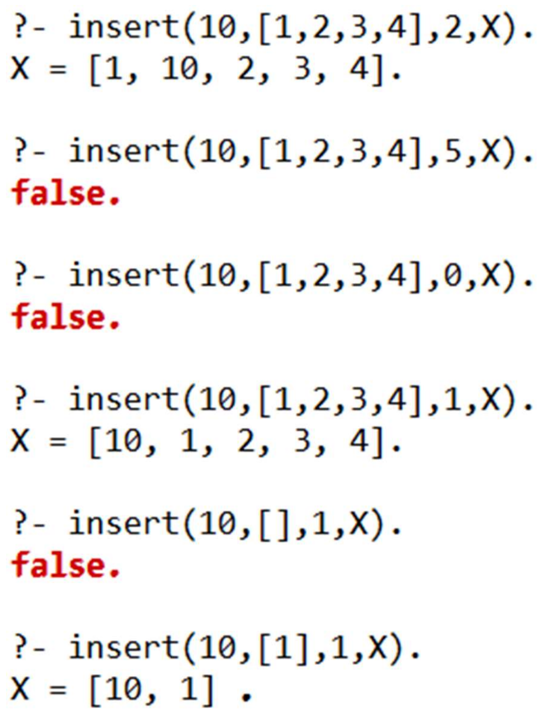
```
?- app_last_conc(16,[],X).
X = [16].
```

Que 8 WAP to insert an element in list at nth position.

Ans

```
insert(I,[],0,[I]).
insert(I,[H|T],AT,X):-
    AT > 0,
    AT=:=1 ->
        prepend(H,T,R1),prepend(I,R1,X);
        AT1 is AT-1,
        insert(I,T,AT1,R2),
        prepend(H,R2,X).
```

#Output



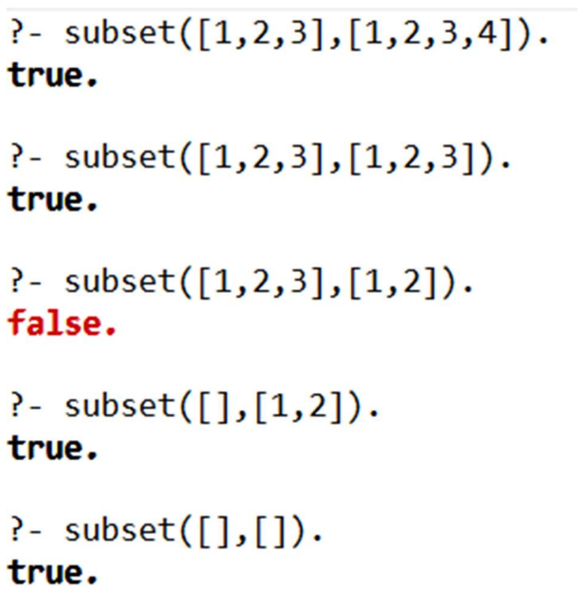
```
?- insert(10,[1,2,3,4],2,X).
X = [1, 10, 2, 3, 4].

?- insert(10,[1,2,3,4],5,X).
false.

?- insert(10,[1,2,3,4],0,X).
false.

?- insert(10,[1,2,3,4],1,X).
X = [10, 1, 2, 3, 4].

?- insert(10,[],1,X).
false.

?- insert(10,[1],1,X).
X = [10, 1] .
```

Que 9 WAP to find if first list is a subset of second list using concat.

Ans

```
subset(L,LR):- concat(_,L2,LR),concat(L,_,L2),!.
```

#Output



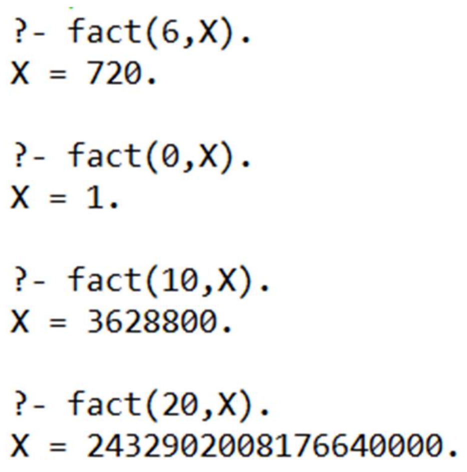
```
?- subset([1,2,3],[1,2,3,4]).
true.

?- subset([1,2,3],[1,2,3]).
true.

?- subset([1,2,3],[1,2]).
false.

?- subset([],[1,2]).
true.

?- subset([],[]).
true.
```

Que 10 WAP to find factorial of a number.

Ans

```
fact(0,1):- !   .
fact(X,R):- X>=0,X1 is X-1,fact(X1,R1),R is X*R1.
```

#Output



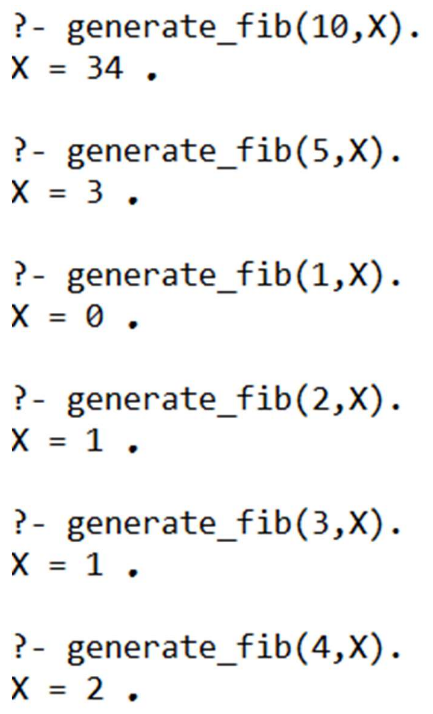
```
?- fact(6,X).
X = 720.

?- fact(0,X).
X = 1.

?- fact(10,X).
X = 3628800.

?- fact(20,X).
X = 2432902008176640000.
```

Que 11 WAP to get Fibonacci series’ nth element.

Ans

```
generate_fib(1,0).
generate_fib(2,1).
generate_fib(N,T):-
    N>2,
    N1 is N-1,
    N2 is N-2,
    generate_fib(N2,T2),
    generate_fib(N1,T1),
    T is T1+T2.
```

#Output



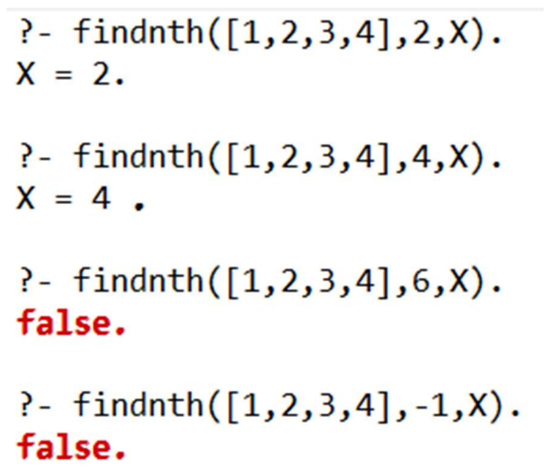
```
?- generate_fib(10,X).
X = 34 .

?- generate_fib(5,X).
X = 3 .

?- generate_fib(1,X).
X = 0 .

?- generate_fib(2,X).
X = 1 .

?- generate_fib(3,X).
X = 1 .

?- generate_fib(4,X).
X = 2 .
```

Que 12 WAP to find nth element of a list.

Ans

```
findnth([H],1,H).
findnth([H|T],AT,X):-
    AT >= 1,
    AT=:=1 ->
        X is H;
        AT1 is AT-1,
        findnth(T,AT1,X1),
        X is X1.
```

#Output



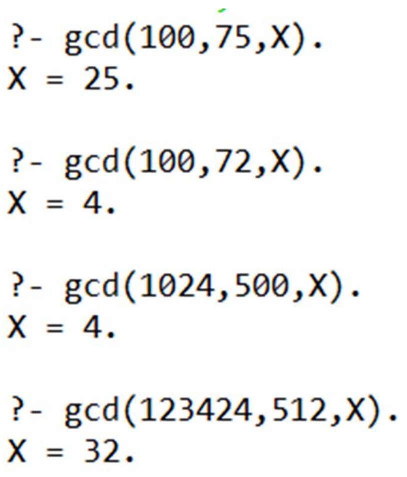
```
?- findnth([1,2,3,4],2,X).
X = 2.

?- findnth([1,2,3,4],4,X).
X = 4 .

?- findnth([1,2,3,4],6,X).
false.

?- findnth([1,2,3,4],-1,X).
false.
```

Que 13 WAP to get greatest common division (GCD) of two numbers.

Ans

```
gcd(0, Y, Y) :- !.
gcd(X, 0, X) :- !.
gcd(X, Y, D) :-
    X > Y ->
        T is X - Y,
        gcd(T, Y, D);
        T is Y - X,
        gcd(T,X,D).
```

#Output



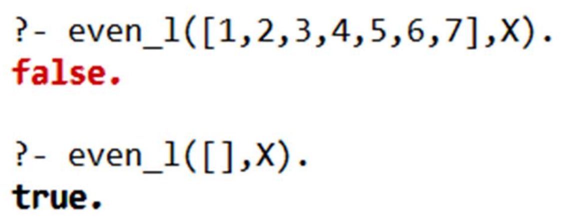
```
?- gcd(100,75,X).
X = 25.

?- gcd(100,72,X).
X = 4.

?- gcd(1024,500,X).
X = 4.

?- gcd(123424,512,X).
X = 32.
```

Que 14 WAP to check if length of list is even.

Ans

```
even_l([],_):- !.
even_l([_|T],R):- odd_l(T,R).
```

#Output



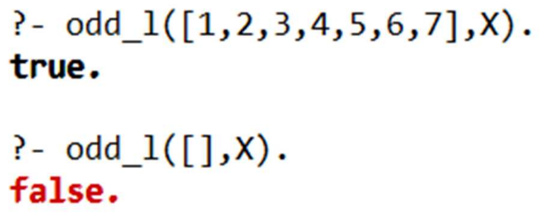
```
?- even_l([1,2,3,4,5,6,7],X).
false.

?- even_l([],X).
true.
```

Que 15 WAP to check if length of list is odd.

Ans

```
odd_l([_],_):- !.
odd_l([_|T],R):- even_l(T,R).
```

#Output



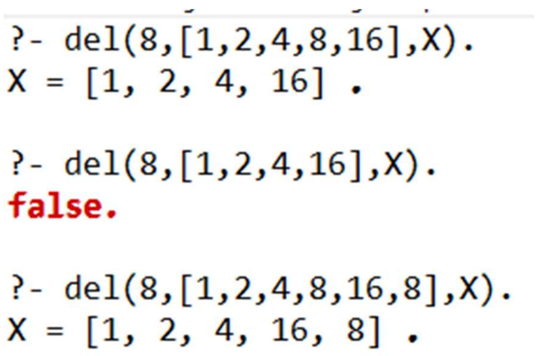
```
?- odd_l([1,2,3,4,5,6,7],X).
true.

?- odd_l([],X).
false.
```

Que 16 WAP to delete given element from list.

Ans

```
del(X,[X|T],T).
del(X,[H|T],[H|T1]):- del(X,T,T1).
```

#Output



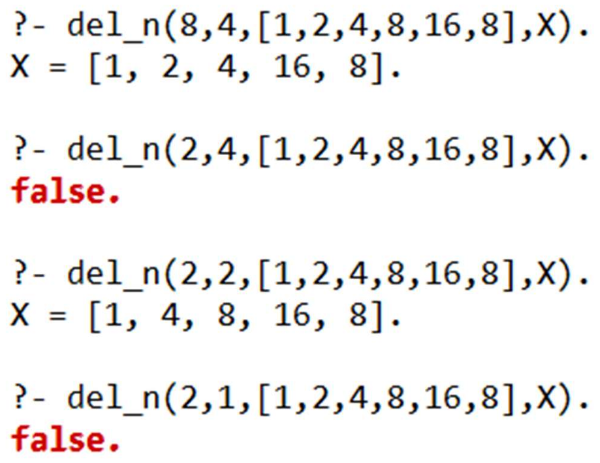
```
?- del(8,[1,2,4,8,16],X).
X = [1, 2, 4, 16] .

?- del(8,[1,2,4,16],X).
false.

?- del(8,[1,2,4,8,16,8],X).
X = [1, 2, 4, 16, 8] .
```

Que 17 WAP to delete nth element from list.

Ans

```
del_n(X,1,[X|T],T):- !.
del_n(X,N,[H|T],[H|R]):- N>1,N1 is N-1,del_n(X,N1,T,R).
```

#Output



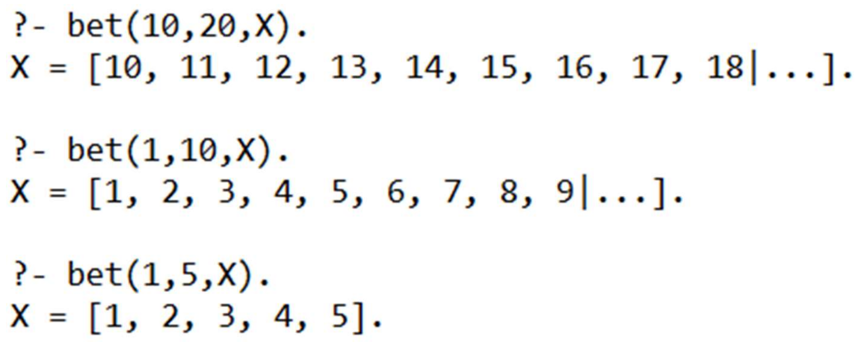
```
?- del_n(8,4,[1,2,4,8,16,8],X).
X = [1, 2, 4, 16, 8].

?- del_n(2,4,[1,2,4,8,16,8],X).
false.

?- del_n(2,2,[1,2,4,8,16,8],X).
X = [1, 4, 8, 16, 8].

?- del_n(2,1,[1,2,4,8,16,8],X).
false.
```

Que 18 WAP to generate all integer between two numbers.

Ans

```
bet(H,H,[H]):-!.
bet(L,H,[L|R]):- L<H,N is L+1,bet(N,H,R).
```

#Output



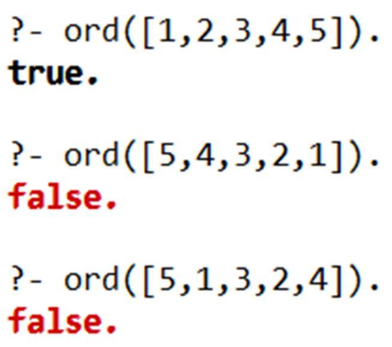
```
?- bet(10,20,X).
X = [10, 11, 12, 13, 14, 15, 16, 17, 18|...].

?- bet(1,10,X).
X = [1, 2, 3, 4, 5, 6, 7, 8, 9|...].

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

Que 19 WAP to check if list is ordered in ascending order or not.

Ans

```
ord([]):-!.
ord([_]):-!.
ord([A,B|T]):- A=<B,ord([B|T]).
```

#Output



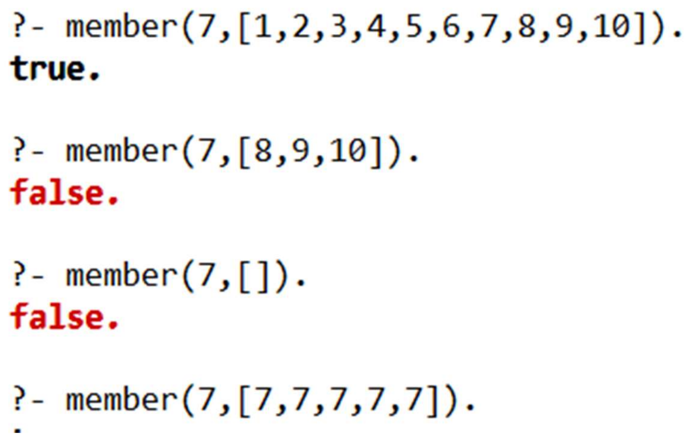
```
?- ord([1,2,3,4,5]).
true.

?- ord([5,4,3,2,1]).
false.

?- ord([5,1,3,2,4]).
false.
```

Que 20 WAP to find a number in list.

Ans

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

#Output



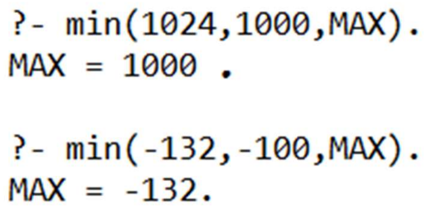
```
?- member(7,[1,2,3,4,5,6,7,8,9,10]).
true.

?- member(7,[8,9,10]).
false.

?- member(7,[]).
false.

?- member(7,[7,7,7,7,7]).
```

Que 21 WAP to find minimum between two numbers.

Ans

```
min(X,Y,MIN):-X>=Y,MIN is Y.
min(X,Y,MIN):- X<Y,MIN is X.
```

#Output



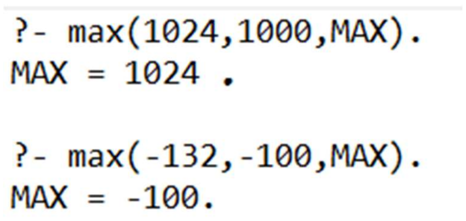
```
?- min(1024,1000,MAX).
MAX = 1000 .

?- min(-132,-100,MAX).
MAX = -132.
```

Que 22 WAP to find maximum between two numbers.

Ans

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

#Output



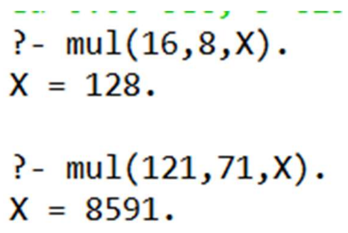
```
?- max(1024,1000,MAX).
MAX = 1024 .

?- max(-132,-100,MAX).
MAX = -100.
```

Que 23 WAP to multiply two numbers.

Ans

```
mul(X,Y,R):- R is X*Y.
```

#Output



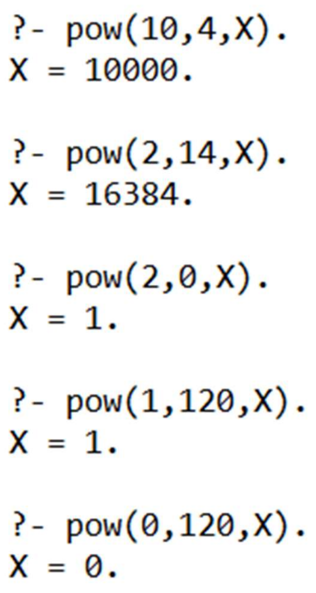
```
?- mul(16,8,X).
X = 128.

?- mul(121,71,X).
X = 8591.
```

Que 24 WAP to find the value of first number raised to the power of second number.

Ans

```
pow(_,0,R):- R is 1,!.
pow(X,N,R):- N>0,N1 is N-1,pow(X,N1,R1),R is R1*X.
```

#Output



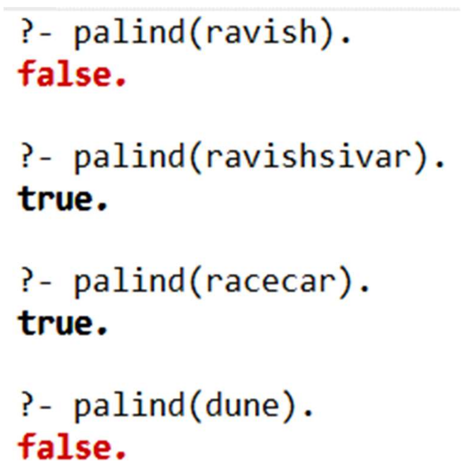
```
?- pow(10,4,X).
X = 10000.

?- pow(2,14,X).
X = 16384.

?- pow(2,0,X).
X = 1.

?- pow(1,120,X).
X = 1.

?- pow(0,120,X).
X = 0.
```

Que 25 WAP to check if a given string is a palindrome or not

Ans

```
app([],X,[X]).
app([H|T1],X,[H|T2]):- app(T1,X,T2).


reverse([],[]).
reverse([X],[X]).
reverse([H|T],A):- reverse(T,R1),app(R1,H,A).

palind(R):-atom_chars(R,R1),reverse(R1,R1),!.
```

#Output



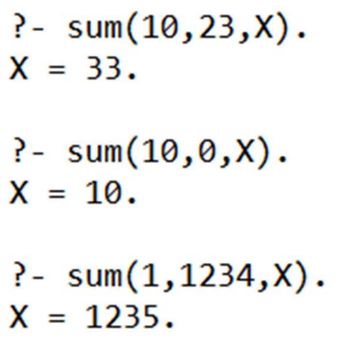
```
?- palind(ravish).
false.

?- palind(ravishsivar).
true.

?- palind(racecar).
true.

?- palind(dune).
false.
```

Que 26 WAP to perform sum of two numbers

Ans

```
sum(N1,N2,R) :- R is N1+N2.
```

#Output



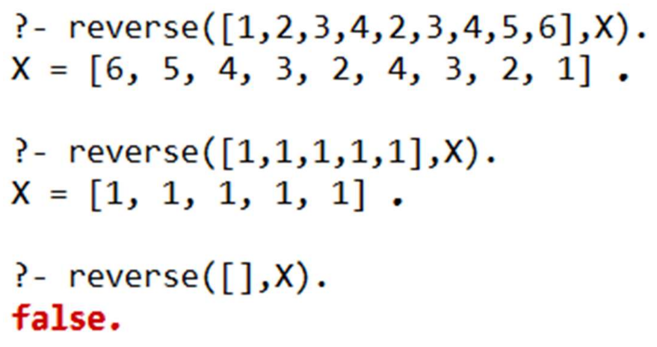
```
?- sum(10,23,X).
X = 33.

?- sum(10,0,X).
X = 10.

?- sum(1,1234,X).
X = 1235.
```

Que 27 WAP to reverse a list

Ans

```
reverse([H],[H]).
reverse([H|T],R) :- reverse(T,R1),append(R1,H,R).
```

#Output



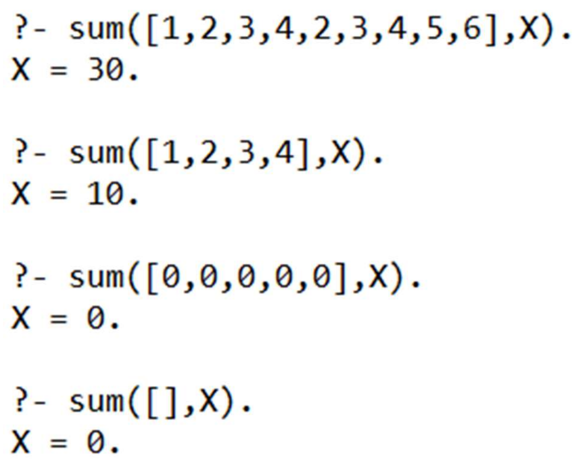
```
?- reverse([1,2,3,4,2,3,4,5,6],X).
X = [6, 5, 4, 3, 2, 4, 3, 2, 1] .

?- reverse([1,1,1,1,1],X).
X = [1, 1, 1, 1, 1] .

?- reverse([],X).
false.
```

Que 28 WAP to calculate sum of a list

Ans

```
sum([],0).
sum([H|T],R):- sum(T,R1), R is H+R1.
```

#Output



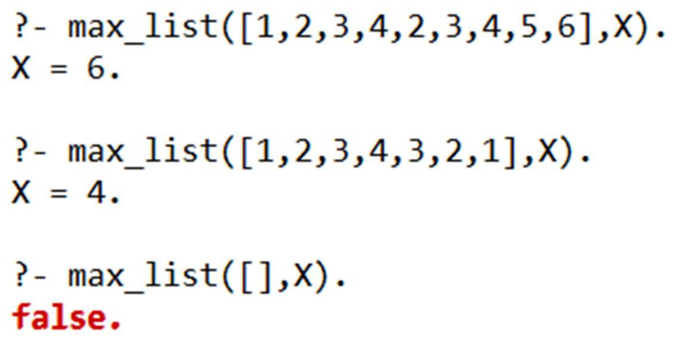
```
?- sum([1,2,3,4,2,3,4,5,6],X).
X = 30.

?- sum([1,2,3,4],X).
X = 10.

?- sum([0,0,0,0,0],X).
X = 0.

?- sum([],X).
X = 0.
```

Que 29 WAP to get max from a list

Ans

```
max_list([H],H).
max_list([H|T],R):- max_list(T,R1), R1>=H ->R is R1;R is H .
```

#Output



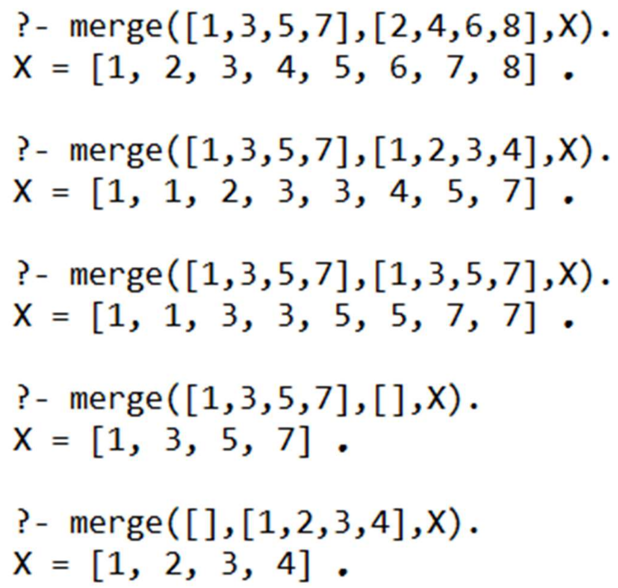
```
?- max_list([1,2,3,4,2,3,4,5,6],X).
X = 6.

?- max_list([1,2,3,4,3,2,1],X).
X = 4.

?- max_list([],X).
false.
```

Que 30 WAP Write a program in PROLOG to implement merge (L1, L2, L3) where L1 is first ordered list and L2 is second ordered list and L3 represents the merged list.

Ans

```
merge([], Ls, Ls).
merge(Rs, [], Rs).
merge([X1 | Xs1], [X2 | Xs2], [X1 | Ms]) :-
  X1 < X2, !,
  merge(Xs1, [X2 | Xs2], Ms).
merge([X1 | Xs1], [X2 | Xs2], [X2 | Ms]) :-
  X1 >= X2, !,
  merge([X1 | Xs1], Xs2, Ms).
merge([X1 | Xs1], [X2 | Xs2], [X1, X2 | Ms]) :-
  X1 = X2, !,
  merge(Xs1, Xs2, Ms).
```

#Output



```
?- merge([1,3,5,7],[2,4,6,8],X).
X = [1, 2, 3, 4, 5, 6, 7, 8] .

?- merge([1,3,5,7],[1,2,3,4],X).
X = [1, 1, 2, 3, 3, 4, 5, 7] .

?- merge([1,3,5,7],[1,3,5,7],X).
X = [1, 1, 3, 3, 5, 5, 7, 7] .

?- merge([1,3,5,7],[],X).
X = [1, 3, 5, 7] .

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

Que 31 WAP to show family relation.

Ans

```
parent(sohan,priya).
parent(priya,bobby).
parent(tilak,bobby).
parent(tilak,lalita).
parent(bobby,anita).
parent(bobby,pratiksha).
parent(pratiksha,john).
male(sohan).
male(tilak).
male(bobby).
male(john).
female(priya).
female(lalita).
female(anita).
female(pratiksha).

child(X,Y):- parent(Y,X).
mother(X,Y):- parent(X,Y),female(X).
father(X,Y):- parent(X,Y),male(X).
```

```
sister(X,Y):- parent(Z,X),parent(Z,Y),X\==Y,female(X).
brother(X,Y):- parent(Z,X),parent(Z,Y),X\==Y,male(X).

grandparent(X,Y):- parent(X,Z),parent(Z,Y).
grandchild(X,Y):- grandparent(Y,X).
grandfather(X,Y):- grandparent(X,Y),male(X).
grandmother(X,Y):- grandparent(X,Y),female(X).

ancestor(X,Y):- parent(X,Y).
ancestor(X,Y):- parent(X,Z),ancestor(Z,Y).
```

#Output

```
?- parent(sohan,X).
X = priya.

?- male(X).
X = sohan ;
X = tilak ;
X = bobby ;
X = john.

?- child(priya,X).
X = sohan.

?- father(X,priya).
X = sohan.

?- grandfather(X,bobby).
X = sohan .

?- ancestor(X,anita).
X = bobby ;
X = sohan ;
X = priya ;
X = tilak ;
false.
```