**1. print all elements of a list ?-print\_list([a,b,c]). a b c**

print\_list([]):-nl. %nl = newline

print\_list([H|T]):-write(H),write(' '),print\_list(T).

**2. reverse all elements of a list ?-reversex([a,b,c],X). X=[c,b,a]**

addtoend(H,[],[H]).

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

reversex([],[]).

reversex([H|T],Y):- reversex(T,T1), addtoend(H,T1,Y).

**3. create list ?-create\_list(5,12,S). S=[5,6,7,8,9,10,11,12]**

create\_list(X,X,[X]).

create\_list(A,X,[A|T]):- AA is A+1, create\_list(AA,X,T).

**4. mean value [1,2,3,4,5] => 3**

sum\_list([],0,0).

sum\_list([H|T],Length,Sum):-sum\_list(T,L1,S1), Length is L1+1, Sum is S1+H.

mean(L,M):-sum\_list(L,Length,Sum), M is Sum/Length.

**5. detect whether list contains a number [a,b,c,d,e,1,f] => T**

numberinlist([]):-fail.

numberinlist([X|T]):-number(X).

numberinlist([X|T]):-numberinlist(T).

**6. increment elements of list [5,6,7,8] => [6,7,8,9]**

increment([],[]).

increment([H|T],[X|Y]):-increment(T,Y),X is H+1.

**7. factorial function**

factorial(0,1).

factorial(N,X):-N>0, N1 is N-1, factorial(N1,S), X is S\*N.

**8. implement append function [a,1,2,b,c], [b,c,d,e] => [a,1,2,b,c,b,c,d,e]**

appendx([],A,A).

appendx([H|T],A,[H|U]):-appendx(T,A,U).

**9. encapsulate list elements [a,b,1,d,e] => [[a],[b],[1],[d],[e]]**

encapsulate([],[]).

encapsulate([H|T],[[H]|Y]):-encapsulate(T,Y).

**10. insert zeros [1,2,3,4,5] => [1,0,2,0,3,0,4,0,5,0]**

insert\_zeros([],[]).

insert\_zeros([H|T],[H,0|Y]):-insert\_zeros(T,Y).

**11. clone list [g,6,7] => [[g,6,7][g,6,7]]**

clone\_list(T,[T,T]).

**12. modify list element ?-modify\_list([m,o,d,i,f,y,e,t], 6, i,Y). Y=[m,o,d,i,f,y,i,t], ?-modify\_list([m,o,d],6,i,Y). Y=[m,o,d]**

modify\_list([],N,X,[]).

modify\_list([H|T],0,X,[X|T]).

modify\_list([H|T],N,X,[H|Y]):-N>0, N1 is N-1, modify\_list(T,N1,X,Y).**13. rotate list [1,2,3,4,5] => [2,3,4,5,1]**

addtoend(H,[],[H]).

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

rotate\_list([H|T],L1):-addtoend(H,T,L1).

**14. Fibonacci numbers: print nth Fibonacci number.**

fib(1,1).

fib(2,1).

fib(N,F):- N>2,

N1 is N-1, fib(N1,F1),

N2 is N-2, fib(N2,F2),

F is F1+F2.

**15. generate random (values 0-9) square matrix: [[2,4,5],[1,0,3],[9,3,2]]. Inner elements represent matrix rows.**

random10(N):-frandom(X), Y is X\*10, fix(Y,N).

rand\_row(0,[]).

rand\_row(N,[H|T]):-N>0,random10(H),N1 is N-1, rand\_row(N1,T).

square\_matrix\_rand(N,S):-smr(N,N,S).

smr(N,0,[]).

smr(N,X,[R|T]):-N>0, rand\_row(N,R),X1 is X-1,smr(N,X1,T).

**16. member ?-memberx(2,[1,2,3]). yes, ?-memberx([1,2,a],[1,2,3,[1,2,a]]). yes, ?-memberx(4,[1,2,3,[1,2,a]]). no**

memberx(N,[N|T]).

memberx(N,[X|T]):-memberx(N,T).

**17. implement insertion into a sorted list (the result is sorted as well)**

insertinto(N,[],[N]).

insertinto(N,[H|T],[N,H|T]):-H>=N,!.

insertinto(N,[H|T],[H|Y]):-insertinto(N,T,Y).

**18. search duplicates. Result don't contain duplicate elements. [a,b,1,c,3,d,2,2,f,3] => [3,2]**

memberx(N,[N|T]).

memberx(N,[X|T]):-memberx(N,T).

deleteall(N,[],[]).

deleteall(N,[N|T],U):-!,deleteall(N,T,U).

deleteall(N,[H|T],[H|U]):-deleteall(N,T,U).

delete\_dupl([],[]).

delete\_dupl([H|T],Y):-memberx(H,T),!,deleteall(H,T,T1),delete\_dupl(T1,Y).

delete\_dupl([H|T],[H|Y]):-delete\_dupl(T,Y).

**19. remove unique elements [6,2,3,3,5,2,3,1,4] => [2,3,3,2,3]**

memberx(N,[N|T]).

memberx(N,[X|T]):-memberx(N,T).

delete\_unique([],[]).

delete\_unique([H|T],[H|Y]):-memberx(H,T),!,delete\_unique(T,Y).

delete\_unique([H|T],Y):-delete\_unique(T,Y).

**20. make a list unique [a,b,c,d,a,b,e,f] => [a,b,c,d,e,f]**

memberx(N,[N|T]).

memberx(N,[X|T]):-memberx(N,T).

deleteall(N,[],[]).

deleteall(N,[N|T],U):-!,deleteall(N,T,U).

deleteall(N,[H|T],[H|U]):-deleteall(N,T,U).

make\_unique([],[]).

make\_unique([H|T],[H|Y]):-memberx(H,T),!,deleteall(H,T,T1),make\_unique(T1,Y).

make\_unique([H|T],[H|Y]):-make\_unique(T,Y).

or [a,b,c,d,a,b,e,f] => [c,d,a,b,e,f]

1. Define a predicate quicksort(L,K) which, given a list of integers L, returns an ordered list K obtained from L with the method of quicksort.
2. Define a predicate merge(L,K,M) which, given two ordered lists of integers L and K, returns an ordered list M containing all the elements of L and K.
3. Define a predicate add\_up\_list(L,K) which, given a list of integers L, returns a list of integers in which each element is the sum of all the elements in L up to the same position. Example:

?- add\_up\_list([1,2,3,4],K).

K = [1,3,6,10];

No

1. Define a predicate sumlist(L,N) which, given a list of integers L, returns the sum N of all the elements of L.
2. Define a predicate occurs(L,N,X) which holds iff X is the element occurring in position N of the list L.
3. Define a predicate length(L,N) which holds iff N is the length of the list L.
4. Define a predicate occurrences(X,L,N) which holds iff the element X occurs N times in the list L.
5. Define a predicate reverse(L,K) which holds if and only if the list K is the reverse of the list L.
6. Write a predicate, nth(N, TheList, TheItem), which is true if TheItem is the N’th item in TheList. Counting begins at one. nth(1,Alist,Elem) is true for the first item in the list.
7. Write simple Prolog functions such as the following. Take into account lists which are too short.

-- remove the N’th item from a list.

-- insert as the N’th item.

1. reverse all elements of a list ?-reversex([a,b,c],X). X=[c,b,a]
2. detect whether list contains a number [a,b,c,d,e,1,f] => T
3. increment elements of list [5,6,7,8] => [6,7,8,9]
4. factorial function
5. implement append function [a,1,2,b,c], [b,c,d,e] => [a,1,2,b,c,b,c,d,e]
6. encapsulate list elements [a,b,1,d,e] => [[a],[b],[1],[d],[e]]
7. insert zeros [1,2,3,4,5] => [1,0,2,0,3,0,4,0,5,0]
8. modify list element ?-modify\_list([m,o,d,i,f,y,e,t], 6, i,Y). Y=[m,o,d,i,f,y,i,t], ?-modify\_list([m,o,d],6,i,Y). Y=[m,o,d]
9. Fibonacci numbers: print nth Fibonacci number.
10. maximum function [1,-2,3] => 3