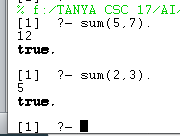
**ARTIFICIAL INTELLIGENCE PROGRAMS**

-Tanya Narang (CSC/17/7)

**Q1. Sum of 2 numbers**

**Source code:**

sum(X,Y):- S is X+Y,write(S).

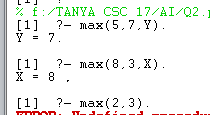
**Output:**

**Q2. Max of 2 numbers**

**Source code:**

max(X,Y,M):- X>Y, M is X.

max(\_,Y,M):- M is Y.

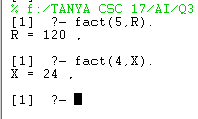
**Output:**

**Q3. Factorial of a number**

**Source code:**

fact(0,1).

fact(N,R):-N1 is N-1, fact(N1,R1), R is N\*R1.

**Output:**

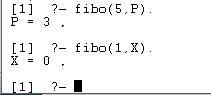
**Q4. Nth term of fibonacci series**

**Source code:**

fibo(1,0).

fibo(2,1).

fibo(P,N):- P1 is P-1,fibo(P1,R1),P2 is P-2,fibo(P2,R2),N is R1+R2.

**Output:**

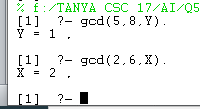
**Q5. GCD of two numbers**

**Source code:**

gcd(X,X,X).

gcd(X,Y,D):- X<Y, gcd(Y,X,D).

gcd(X,Y,D):- X>Y,Y1 is X-Y,gcd(Y,Y1,D).

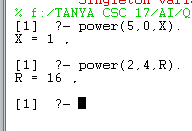
**Output:**

**Q6. Power of a number**

**Source code:**

power(Num,0,Ans):-Ans is 1.

power(Num,Pow,Ans):-P is Pow-1,power(Num,P,Ans1),Ans is Ans1\*Num.

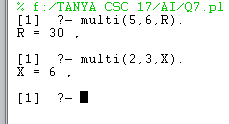
**Output:**

**Q7. Multiplication of two numbers**

**Source code:**

multi(N1,1,N1).

multi(N1,N2,Ans):- Temp is N2-1, multi(N1,Temp,Ans1),Ans is Ans1+N1.

**Output:**

**Q8. Tower of Hanoi.**

**Source code:**

mov(0,\_,\_,\_):-!.

mov(N,L,C,R):- N1 is N-1,mov(N1,L,R,C),write("Move disk "),write(N),write(" from "),write(L),write(" to "),write(C),nl,mov(N1,C,L,R).

toh(N):-mov(N,left,center,right).

%L is treated as Source,R as destination and C as intermediate peg.

**Output:**

**Q9. Cyclic directed graph.**

**Source code:**

edge(p,q).

%edge(q,r).

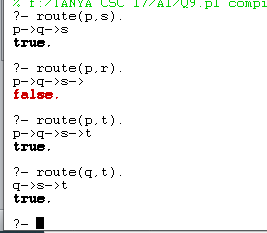
edge(q,s).

edge(s,t).

route(X,X):-write(X),!.

%route(X,Y):-edge(X,Y),write(X),write("->"),write(Y),!.

route(X,Y):-edge(X,Z),write(X),write("->"),route(Z,Y).



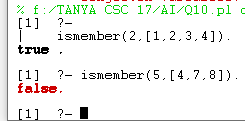
**Output:**

**Q10. X is a member of list or not.**

**Source code:**

ismember(X,[X|T]).

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

**Output:**

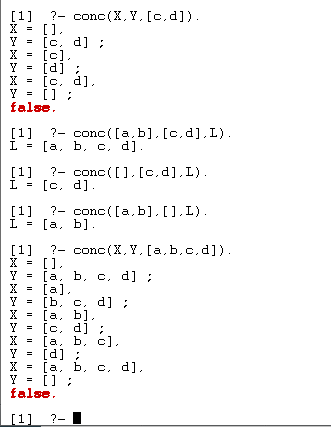
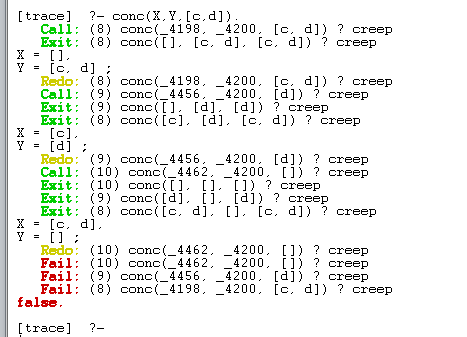
**Q11. Concatenate 2 lists and store in 3rd list.**

**Source code:**

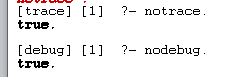
conc([],L,L).

conc([H|T],L2,[H|L3]):-conc(T,L2,L3).

**Output:**



Commands to go back from trace.



**Q12. Reverse a list.**

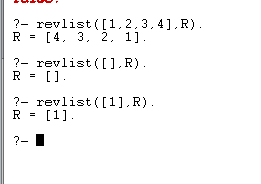
**Source code:**

conc([],L,L).

conc([H|T],L2,[H|L3]):-conc(T,L2,L3).

revlist([],[]).

revlist([H|T],R):-revlist(T,Trev),conc(Trev,[H],R).

**Output:**

**Q13. Check whether a list is palindrome or not.**

**Source code:**

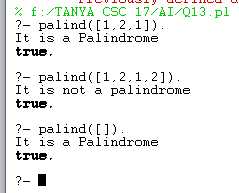
conc([],L,L).

conc([H|T],L2,[H|L3]):-conc(T,L2,L3).

revlist([],[]).

revlist([H|T],R):-revlist(T,Trev),conc(Trev,[H],R).

palind(L):-revlist(L,R), L=R -> write("It is a Palindrome");write("It is not a palindrome").

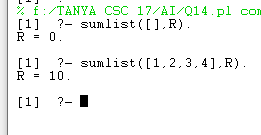
**Output:**

**Q14. Sum of elements of given list.**

**Source code:**

sumlist([],0).

sumlist([H\T],S):-sumlist(T,S1),S is H+S1.

**Output:**

**Q15. Even length or odd length list.**

**Source Code:**

len([],0).

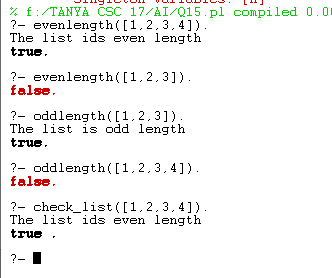
len([H|T],N) :- len(T,N1), N is N1+1.

evenlength(List):- len(List,N),mod(N,2)=:=0,write("The list ids even length").

oddlength(List):-len(List,N),mod(N,2)=\=0,write("The list is odd length").

check\_list(L):- evenlength(L);oddlength(L).

/\* ; mean either goal can be true \*/



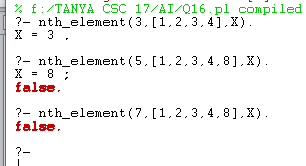
**Output:**

**Q16. Nth element in a list.**

**Source Code:**

nth\_element(1,[H|T],H).

nth\_element(N,[H|T],X):-N1 is N-1, nth\_element(N1,T,X).

**Output:**

**Q17. Write a Prolog program to implement remdup( L, R) to remove duplicates from a list L to generate a list R.**

**Source code:**

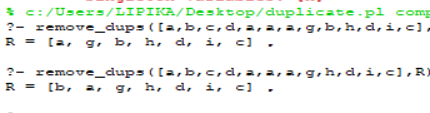
ismember(X,[X|T]).

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

remove\_dups([],[]).

remove\_dups([H|T],R):-ismember(H,T),remove\_dups(T,R).

remove\_dups([H|T],[H|R]):-not(ismember(H,T)),remove\_dups(T,R).

**Output:**

**Q18. Find maximum element in list.**

**Source code:**

max(X,Y,M) :- (X=Y, M is X;

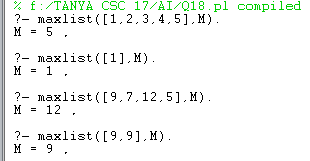
X>Y , M is X;

M is Y

).

maxlist([H|[]], H).

maxlist([H|T],M) :- maxlist(T,M1), max(H,M1,M).

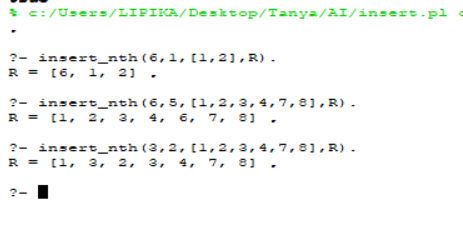
**Output:** 

**Q19. Write a prolog program to implement insert\_nth(I, N, L, R) that inserts an item I into Nth position of list L to generate a list R.**

**Source code :**

insert\_nth(I, 1, L, [I|L]).

insert\_nth(I, N, [H|T], [H|R]):- N1 is N-1,insert\_nth(I, N1, T, R).

**Output :** 

**Q20.** [**Write a Program in PROLOG to implement sublist(S, L) that checks whether the list S is the sublist of list L or not. (Check for sequence or the part in the same order).**](https://www.tutorialsduniya.com/programs/du/csh/artificial-intelligence/program20)

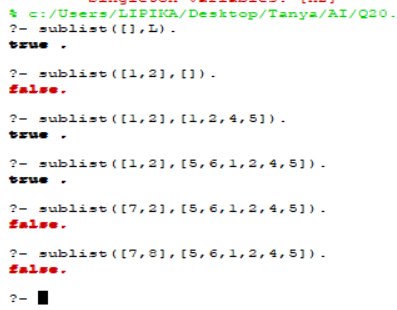
**Source code:**

sublist([],L).

sublist(S,[]):- false.

sublist([H1|T1],[H1|T2]):- sublist(T1,T2).

sublist([H1|T1],[H2|T2]):- sublist([H1|T1],T2).

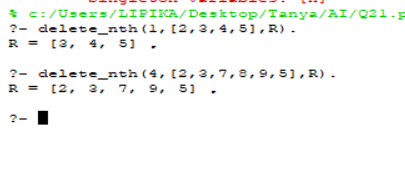
**Output:**

**Q21. :**[**Write a Prolog program to implement delete\_nth (N, L, R) that removes the element on Nth position from a list L to generate a list R.**](https://www.tutorialsduniya.com/programs/du/csh/artificial-intelligence/program21)

**Source code:**

delete\_nth(1, [H|T], T).

delete\_nth(N, [H|T], [H|R]):- N1 is N-1,delete\_nth(N1, T, R).

**Output:**

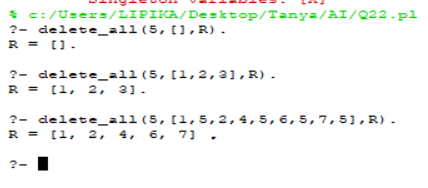
**Q22. Write a program in PROLOG to implement delete\_all (X, L, R) where X denotes the element whose all occurrences has to be deleted from list L to obtain list R.**

**Source code:**

delete\_all(X, [], []).

delete\_all(X, [X|T], R):- delete\_all(X, T, R).

delete\_all(X, [H|T], [H|R]):- delete\_all(X, T, R).

**Output:**

**Q23. 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.**

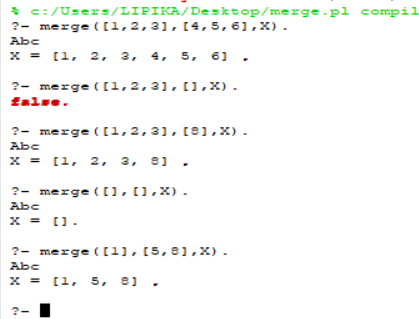
**Source code:**

merge([],L,L):-!,write("Abc").

merge([H1|T1],[H2|T2],[H3|T3]):-H1>H2,H3 is H2,merge([H1|T1],T2,T3).

merge([H1|T1],[H2|T2],[H3|T3]):-H1<H2,H3 is H1,merge(T1,[H2|T2],T3).

merge([H1|T1],[H2|T2],[H3,H4|T3]):-H1=H2,H3 is H1,H4 is H2,merge(T1,T2,T3).



**Output:**

**Q24. Write a PROLOG program that will take grammar rules in the following format:  
          NT -> (NT | T)\*  
Where NT is any nonterminal, T is any terminal and Kleene star (\*) signifies any number of repetitions, and generate the corresponding top-down parser, that is:  
        sentence -> noun-phrase, verb-phrase  
        determiner -> [the]will generate the following:  
        sentence (I, O) :- noun-phrase(I,R), verb-phrase (R,O).  
        determiner ([the|X], X) :- !.**

**Source code:**

sentence-->np,vp.

np-->det,noun.

vp-->verb.

vp-->verb,np.

det-->[a].

det-->[the].

det-->[an].

noun-->[boy].

noun-->[girl].

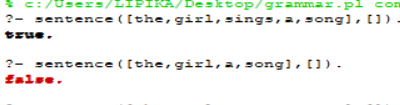
noun-->[song].

noun-->[apple].

verb-->[sing].

verb-->[sings].

verb-->[eats].

**Output:**

**Q25. Write a prolog program that implements Semantic Networks.**

**Source code:**

%Facts%

mat(mat1).

sat\_on(Cat1,mat1).

cat(Cat1).

cat(tom).

color(tom,black).

owns(john,tom).

bird(bird1).

caught(tom,bird1).

is\_coloured(tom,ginger).

%Rules%

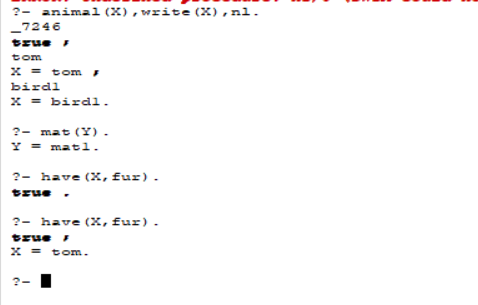
animal(X):- mammal(X).

animal(X):- bird(X).

mammal(X):- cat(X).

have(X,fur):- mammal(X).

likes(X,cream):- cat(X).

**Output:** 

**Source code:**

**(b)**

ako(mammal,animal).

ako(bird,mammal).

ako(elephant,mammal).

is\_a(tweety,bird).

is\_a(clyde,elephant).

is\_a(neil,elephant).

can(animal,breathe).

can(animal,move).

can(bird,fly).

can(tweety,sing).

has(neil,three\_legs).

has(bird,feather).

has(bird,wing).

has(elephant,four\_legs).

has(elephant,one\_trunk).

has(elephant,one\_tail).

has(animal,skin).

has(mammal,head).

color(elephant,grey).

color(tweety,yellow).

like(neil,apple).

%Rules

check(O,P,V):- G=..[P,O,V],G,!;!.

check(O,P,V):-is\_a(O,X),check(X,P,V);ako(O,X),check(X,P,V).

**Output:**