# CS4271 Artificial Intelligence Lab

## Assignment 1

Name: Abhiroop Mukherjee
Enrolment No.:510519109
GSuite ID: 510519109.abhirup@students.iiests.ac.

## Entire Code

```
% 1. To determine whether the first two elements of a list
% are same
first_two_same([X|[X|_]]).

% 2. To determine whether a list is not a two-element list.
not_two_element_list([]).
not_two_element_list([_]).
not_two_element_list([_| [_|[_ | _]]]).

% 3. To determine whether two lists are of same length.
equal_length([],[]).
equal_length([_ | L1], [_ | L2]):-
    equal_length(L1, L2).

% 4. To determine length of a list using your own number system.
% length 0 -> 0
% length 1 -> s(0)
% length 2 -> s(s(0))
my_list_length([], 0).
my_list_length([_| Rest], s(L)):-
    my_list_length(Rest, L).

% 5. To determine whether two lists are of same length using
% the length predicate developed in 4 (previous problem).
equal_length1(L1, L2):-
    my_list_length(L1, X),
    my_list_length(L2, X).

% 6. To find the last element of a list.
last_element([X], X).
last_element([_ | [B |L1]], X):-
    last_element([B | L1], X).

% 7. To find whether an element is a member of a list.
```

```
is_member(X, [X]).
is_member(X, [_ | Rest]):-
    is_member(X, Rest).

% 8. To find whether two elements are next to each other in
% a list.
two_elem_next(A, B, [A | [B | _]]):- !.
two_elem_next(A, B, [_ | Rest]):-
    two_elem_next(A, B, Rest).

% 9. To append two lists in a third list.
% append_two_list(L1, L2, L3). -> appends L1 and L2 and puts it in L3
append_two_list([], L2, L2).
append_two_list([X|L1], L2, [X|L3]):-
    append_two_list(L1, L2, L3).

% 10. To find the last element of a list using append
% predicate developed in 9
last_element1(L1, X):-
    append_two_list(_, [X], L1).

% 11. To find whether an element is a member of a list using
% append predicate developed in 9
% is_member(X, L1). -> true if X is in L1
is_member1(X, L1):-
    append_two_list(_, [X|_], L1).

% 12. To find whether two elements are next to each other in a list
using append predicate
% developed in 9.
% next_each_other(X, Y, L)
next_each_other(X, Y, L):-
    append_two_list(_, [X|[Y|_]], L).

% 13. To reverse a list in another list.
% reverse_list(L, R), R is the resultant
reverse_list([], []).
reverse_list([X|L], L1):-
    reverse_list(L, L2),
    append_two_list(L2, [X], L1).

% 14. To determine whether a list is a palindrome.
% is_palindrome(L) -> give true if palindrome
is_palindrome(L):-
    reverse_list(L, L).
```

```
% 15. To find the last but one element of a list.
% last_but_one(L, X) -> X is the last but one element of list L
last_but_one(L, X):-
    append_two_list(_, [X|[_]], L).

% 16. To find the sum of all elements of a list.
% sum_list(L, S) -> S is the sum of all elements of list L
sum_list([], 0).
sum_list([H|T], Sum) :-
   sum_list(T, Rest),
   Sum is H + Rest.

% 17. To find the maximum number from a list.

% max_two_num(X, Y, Z) -> Z is the max number between X and Y
max_two_num(X, Y, X):-
    X >= Y.
max_two_num(X, Y, Y):-
    X =< Y.

% max_list(L, X) -> X is the maximum number of list L
max_list([X], X).
max_list([X | Rest], Max):-
    max_list(Rest, MaxRest),
    max_two_num(X, MaxRest, Max).

% 18. To find gcd of two integers.
% gcd_two_num(X, Y, Z) -> Z is the gcd of X and Y
gcd_two_num(X,Y,G) :- X=Y, G=X.
gcd_two_num(X,Y,G) :- X<Y, Y1 is Y-X, gcd_two_num(X,Y1,G).
gcd_two_num(X,Y,G) :- X>Y ,gcd_two_num(Y,X,G).

% 19. To determine whether a given integer number is prime.
% divisible(X, Y), X is divisible by Y
divisible(X,Y) :- 0 is X mod Y, !.
divisible(X,Y) :- X > Y+1, divisible(X, Y+1).
% is_prime(X) -> true if X is prime
is_prime(2) :- true,!.
is_prime(X) :- X < 2,!,false.
is_prime(X) :- not(divisible(X, 2)).

% 20. To determine whether two positive integer numbers are co-prime.
% coprime(X, Y) -> true if X and Y are coprime
coprime(X, Y):-
    gcd_two_num(X, Y, 1).
```

```
% 21. To determine the prime factors of a given positive integer.
prime_factors(N, L) :-
    findall(D, prime_factor(N, D), L).

prime_factor(N, D) :-
    find_prime_factor(N, 2, D).

find_prime_factor(N, D, D) :-
    0 is N mod D.
find_prime_factor(N, D, R) :-
    D < N,
    (0 is N mod D
    -> (N1 is N/D, find_prime_factor(N1, D, R))
    ;  (D1 is D + 1, find_prime_factor(N, D1, R))
    ).

% 22. Goldbach's conjecture.
% goldbach(X, L)-> L is a list of two prime numbers whose sum is X
next_prime(P,P1) :- P1 is P + 2, is_prime(P1), !.
next_prime(P,P1) :- P2 is P + 2, next_prime(P2,P1).

goldbach(4,[2,2]).
goldbach(N,L) :-
    N mod 2 =:= 0,
    N > 4,
    goldbach(N,L,3).
goldbach(N,[P,Q],P) :-
    Q is N - P,
    is_prime(Q), P < Q.
goldbach(N,L,P) :-
    P < N,
    next_prime(P,P1),
    goldbach(N,L,P1).

% 23. To count numbers greater than 100.0 in a list.
% num_greater_than_100(L, X) -> X are the number of terms
% in list L that are greater than 100

num_greater_than_100([], 0).

num_greater_than_100([Head| Rest], X):-
    Head > 100, !,
    num_greater_than_100(Rest, Y),
    X is Y + 1.

num_greater_than_100([_| Rest], X):-
```

```
    num_greater_than_100(Rest, X).

% 24. To split a list of numbers in two lists such that
% one contains negative numbers and other contains positive numbers.
% split(L, X, Y) -> split L into X have +ve and Y havign -ve
split([],[],[]).

split([Head|Tail], [Head|L1], L2):-
    Head >= 0, !,
    split(Tail, L1, L2).

split([Head|Tail], L1, [Head|L2]):-
    split(Tail, L1, L2).
```

## Output

1. To determine whether the first two elements of a list are same.

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

?- first_two_same([1,1,3,4]).
true.

?- first_two_same([2,1,1,4]).
false.
```

2. To determine whether a list is not a two-element list.

```
?- not_two_element_list([1]).
true .

?- not_two_element_list([1,2]).
false.

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

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

3. To determine whether two lists are of same length.

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

?- equal_length([2, 5], [1,2]).
true.

?- equal_length([2, 5], [1,2,100]).
false.
```

4. To determine length of a list using your own number system.

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

?- my_list_length([1], X).
X = s(0).

?- my_list_length([1,2,3], X).
X = s(s(s(0))).
```

5. To determine whether two lists are of same length using the length predicate developed in 4 (previous problem).

```
?- equal_length([1], [4,5]).
false.

?- equal_length([1,100], [4,5]).
true.

?- equal_length([1,100], [4,5,400]).
false.
```

6. To find the last element of a list.

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

?- last_element([1,100,3], X).
X = 3 .

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

7. To find whether an element is a member of a list.

```
?- is_member(1, [100,200]).
false.

?- is_member(1, [100,200, 1]).
true .

?- is_member(1, [100,200, 1, 50000]).
false.
```

8. To find whether two elements are next to each other in a list.

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

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

?- two_elem_next(1,2,[2,1,2,1,2]).
true.
```

9. To append two lists in a third list.

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

?- append_two_list([],[100],X).
X = [100].

?- append_two_list([300],[100, 400],X).
X = [300, 100, 400].
```

10. To find the last element of a list using append predicate developed in 9

```
?- last_element1([], X).
false.

?- last_element1([1], X).
X = 1 .

?- last_element1([1,100], X).
X = 100 .

?- last_element1([1,100,2], X).
X = 2 .
```

11. To find whether an element is a member of a list using append predicate developed in 9

```
?- is_member1(1, []).
false.

?- is_member1(1, [1000]).
false.

?- is_member1(1, [1000, 1]).
true .

?- is_member1(1, [1000, 1,2]).
true .
```

12. To find whether two elements are next to each other in a list using append predicate developed in 9.

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

?- next_each_other(1,2, [100, 2, 1]).
false.

?- next_each_other(1,2, [100, 2, 1, 2]).
true .

?- next_each_other(1,2, [100, 2, 1, 2, 3]).
true .
```

13. To reverse a list in another list.

```
?- reverse_list([1,2],X).
X = [2, 1].

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

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

?- reverse_list([],X).
X = [].
```

14. To determine whether a list is a palindrome.

```
?- is_palindrome([]).
true.

?- is_palindrome([1]).
true.

?- is_palindrome([1,2]).
false.

?- is_palindrome([1,2,1]).
true.
```

15. To find the last but one element of a list.

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

?- last_but_one([1], X).
false.

?- last_but_one([1,2], X).
X = 1 .

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

16. To find the sum of all elements of a list.

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

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

?- sum_list([1,100], X).
X = 101.

?- sum_list([1,100,5], X).
X = 106.
```

17. To find the maximum number from a list.

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

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

?- max_list([1,100], X).
X = 100 .

?- max_list([1,100,2], X).
X = 100 .
```

18. To find gcd of two integers.

```
?- gcd_two_num(100,5, X).
X = 5 .

?- gcd_two_num(7,5, X).
X = 1 .

?- gcd_two_num(36,12, X).
X = 12 .

?- gcd_two_num(36,18, X).
X = 18 .

?- gcd_two_num(42,18, X).
X = 6 .
```

19. To determine whether a given integer number is prime.

```
?- is_prime(1).
false.

?- is_prime(2).
true.

?- is_prime(100).
false.

?- is_prime(101).
true.
```

20. To determine whether two positive integer numbers are co-prime.

```
?- coprime(100, 102).
false.

?- coprime(1001, 1000001).
true .

?- coprime(20, 100).
false.
```

21. To determine the prime factors of a given positive integer.

```
?- prime_factors(100, X).
X = [2, 2, 5, 5].

?- prime_factors(100001, X).
X = [11, 9091].

?- prime_factors(10000001, X).
X = [11, 909091].

?- prime_factors(12321, X).
X = [3, 3, 37, 37].
```

22. Goldbach's conjecture.

```
?- goldbach(2002, X).
X = [3, 1999] .

?- goldbach(202, X).
X = [3, 199] .

?- goldbach(102, X).
X = [5, 97] .
```

23. To count numbers greater than 100.0 in a list.

```
?- num_greater_than_100([], X).
X = 0.

?- num_greater_than_100([100], X).
X = 0.

?- num_greater_than_100([100, 101], X).
X = 1.

?- num_greater_than_100([100, 101, 1000, 1, 2], X).
X = 2.
```

24. To split a list of numbers in two lists such that one contains negative numbers and other contains positive numbers.

```
?- split([1,-1], X, Y).
X = [1],
Y = [-1].

?- split([1,-1, 0, 100], X, Y).
X = [1, 0, 100],
Y = [-1].

?- split([1,-1, 0, 100, -1.6262], X, Y).
X = [1, 0, 100],
Y = [-1, -1.6262].
```