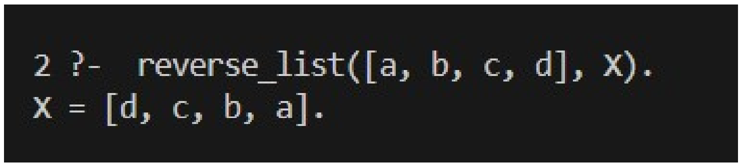
Prolog Assignment Solutions

1. Reverse a List

Knowledge Base:

```
reverse_list([], []).
reverse_list([H|T], R) :- reverse_list(T, RevT), append(RevT
    , [H], R).
```

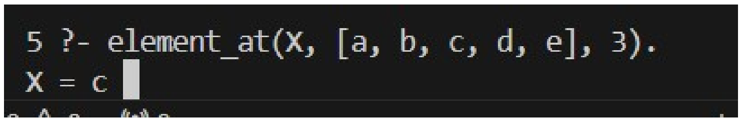


```
2 ?-  reverse_list([a, b, c, d], X).
X = [d, c, b, a].
```

2. Find the K'th Element of a List

Knowledge Base:

```
element_at(X, [X|_], 1).
element_at(X, [_|T], K) :-
    K > 1,
    K1 is K - 1,
    element_at(X, T, K1).
```



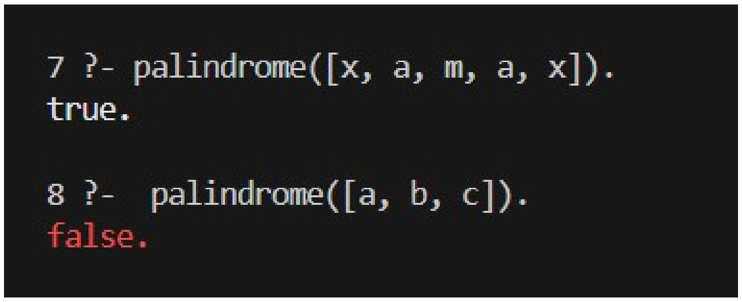
```
5 ?- element_at(X, [a, b, c, d, e], 3).
X = c
```

3. Check if a List is a Palindrome

Knowledge Base:

```
reverse_list([], []).
reverse_list([H|T], R) :- reverse_list(T, RevT), append(RevT
    , [H], R).

palindrome(L) :- reverse_list(L, L).
```



```
7 ?- palindrome([x, a, m, a, x]).
true.

8 ?-  palindrome([a, b, c]).
false.
```

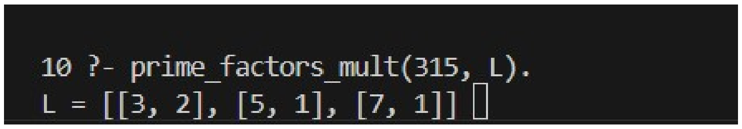
4. Prime Factors with Multiplicity

Knowledge Base:

```
prime_factors_mult(N, L) :- prime_factors(N, L, 2).

prime_factors(1, [], _) :- !.
prime_factors(N, [[F, Count]|T], F) :-
    divisible_count(N, F, Count, R),
    Count > 0,
    prime_factors(R, T, F).
prime_factors(N, L, F) :-
    F1 is F + 1,
    prime_factors(N, L, F1).

divisible_count(N, F, 0, N) :- N mod F =\= 0, !.
divisible_count(N, F, Count, R) :-
    N mod F =:= 0,
    N1 is N // F,
    divisible_count(N1, F, Count1, R),
    Count is Count1 + 1.
```



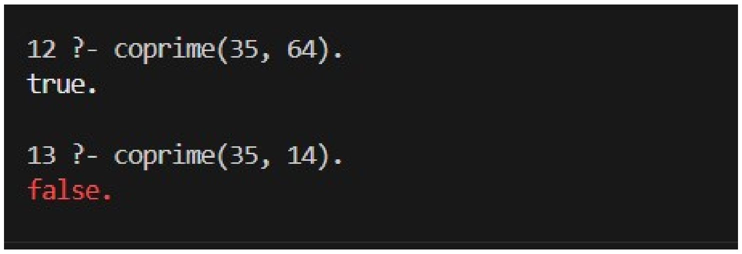
```
10 ?- prime_factors_mult(315, L).
L = [[3, 2], [5, 1], [7, 1]]
```

5. Check if Two Numbers Are Coprime

Knowledge Base:

```
coprime(A, B) :- gcd(A, B, 1).

gcd(X, 0, X) :- !.
gcd(X, Y, G) :- Y > 0, R is X mod Y, gcd(Y, R, G).
```



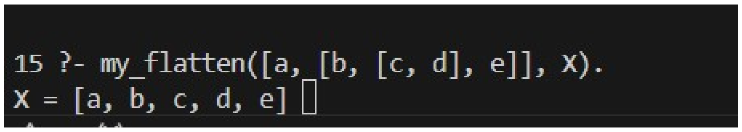
```
12 ?- coprime(35, 64).
true.

13 ?- coprime(35, 14).
false.
```

6. Flatten a Nested List

Knowledge Base:

```
my_flatten([], []).
my_flatten([H|T], Flat) :-
    my_flatten(H, FlatH),
    my_flatten(T, FlatT),
    append(FlatH, FlatT, Flat).
my_flatten(X, [X]) :- \+ is_list(X).
```

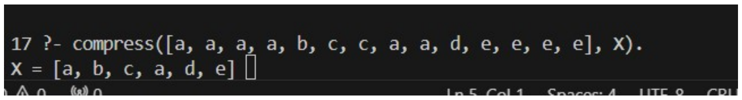


```
15 ?- my_flatten([a, [b, [c, d], e]], X).
X = [a, b, c, d, e]
```

7. Eliminate Consecutive Duplicates

Knowledge Base:

```
compress([], []).
compress([X], [X]).
compress([X, X|T], R) :- compress([X|T], R).
compress([X, Y|T], [X|R]) :- X \= Y, compress([Y|T], R).
```

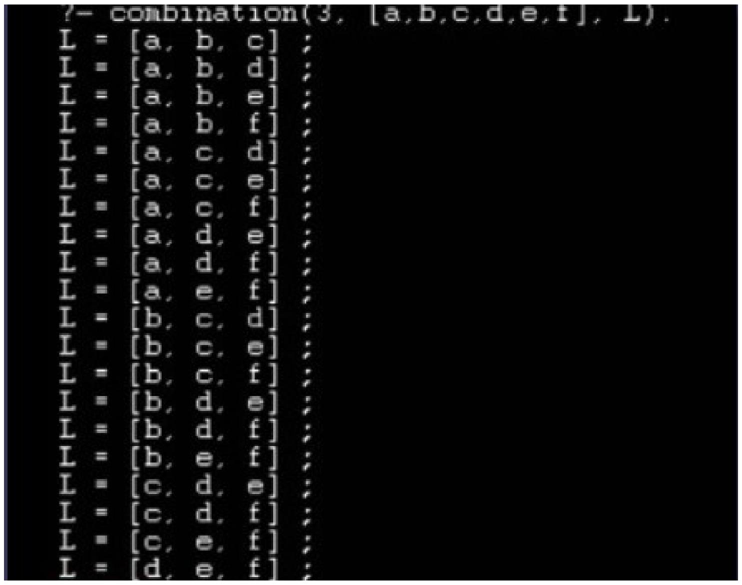


```
17 ?- compress([a, a, a, a, b, c, c, a, a, d, e, e, e, e], X).
X = [a, b, c, a, d, e]
```

8. Generate Combinations

Knowledge Base:

```
combination(0, _, []).
combination(K, [H|T], [H|CombT]) :-
    K > 0,
    K1 is K - 1,
    combination(K1, T, CombT).
combination(K, [_|T], Comb) :-
    K > 0,
    combination(K, T, Comb).
```

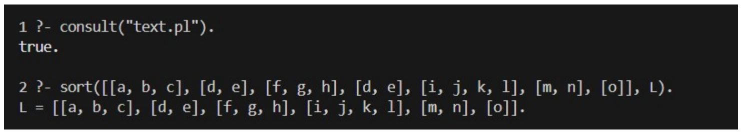


```
?- combination(3, [a,b,c,d,e,f], L).
L = [a, b, c] ;
L = [a, b, d] ;
L = [a, b, e] ;
L = [a, b, f] ;
L = [a, c, d] ;
L = [a, c, e] ;
L = [a, c, f] ;
L = [a, d, e] ;
L = [a, d, f] ;
L = [a, e, f] ;
L = [b, c, d] ;
L = [b, c, e] ;
L = [b, c, f] ;
L = [b, d, e] ;
L = [b, d, f] ;
L = [b, e, f] ;
L = [c, d, e] ;
L = [c, d, f] ;
L = [c, e, f] ;
L = [d, e, f] ;
```

9. Sort List of Lists by Length

Knowledge Base:

```
lsort(InList, OutList) :-
    map_list_to_pairs(length, InList, Pairs),
    keysort(Pairs, SortedPairs),
    pairs_values(SortedPairs, OutList).
```



```
1 ?- consult("text.pl").
true.

2 ?- sort([[a, b, c], [d, e], [f, g, h], [d, e], [i, j, k, l], [m, n], [o]], L).
L = [[a, b, c], [d, e], [f, g, h], [i, j, k, l], [m, n], [o]].
```

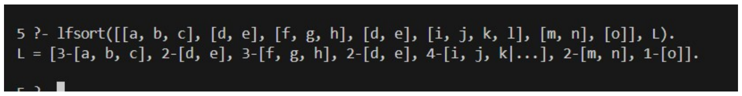
10. Sort List of Lists by Length Frequency

Knowledge Base:

```
lfsort(InList, OutList) :-
    map_list_to_pairs(length, InList, LengthPairs),
    findall(L, member(_-L, LengthPairs), Lengths),
    frequency_sort(LengthPairs, Lengths, OutList).

frequency_sort(Pairs, Lengths, Sorted) :-
    map_list_to_pairs(frequency_count(Lengths), Pairs,
        FrequencyPairs),
    keysort(FrequencyPairs, SortedPairs),
    pairs_values(SortedPairs, Sorted).

frequency_count(Lengths, Len, Freq) :-
    include(=(Len), Lengths, Matches),
    length(Matches, Freq).
```



```
5 ?- lfsort([[a, b, c], [d, e], [f, g, h], [d, e], [i, j, k, l], [m, n], [o]], L).
L = [3-[a, b, c], 2-[d, e], 3-[f, g, h], 2-[d, e], 4-[i, j, k|...], 2-[m, n], 1-[o]].
```