1.

a. bike(wn_entrance,mb_entrance,1).
bike(botanic_garden,wn_entrance,4).
walk(wn_entrance,wn_k137,2).
walk(mb_entrance,mb_6th_floor,5).
route_by_bike(A, B):- bike(A, B, _).
route_by_bike(A, B):- bike(A, C, _), route_by_bike(C, B).

b. route(A, B):- bike(A, B, _); walk(A, B, _).
route(A, B):- bike(A, C, _), route(C, B); walk(A, C, _), route(C, B).

c. routeAcc(A, B, T, R):- bike(A, B, G), R is G+T; walk(A, B, G), R is G+T.
routeAcc(A, B, T, L):- bike(A, C, M), N is M + T, routeAcc(C, B, N, L); walk(A, C, M), N is M + T, routeAcc(C, B, N).

route(A, B, L):- routeAcc(A, B, 0, L).

2.

a. above(X,Y).
X = b,
Y = a ;
X = c,
Y = b ;
X = c,
Y = a ;
b. above_acc(X, Y, A, A):- on(X, Y).
above_acc(X, Y, A, L):- Anew is A+1, on(X, Z), above_acc(Z, Y, Anew, L).

atLeastThree(X):- above_acc(X, Y, 1, L), L >= 3.

c. above_acc(X, Y, A, A):- on(X, Y).
above_acc(X, Y, A, L):- Anew is A+1, on(X, Z), above_acc(Z, Y, Anew, L).

atLeastThree(X):- above_acc(X, Y, 1, L), L =:= 3.
d. acc_tower([X], A, A):- on(X, table).
e. acc_tower([X,Y|T], A, L):- Anew is A+1, on(X, Y), acc_tower([Y|T], Anew, L).

exactlyThreeTower(X):- acc_tower([X,Y|T], 0, L), L =:= 3.

3.

sThe predicate `**xyz/3**` checks whether a list segment defined by its second and third arguments exists within the first argument list. It employs the auxiliary predicate `**xy/4**` to traverse the main list, comparing elements and managing positions through accumulator

lists. When the elements of the sublist are found in sequence in the main list, ending at the point where the sublist is exhausted, the predicate succeeds, indicating the sublist's presence.

4.
```
sum_list([], 0).
sum_list([H|T], Sum):- sum_list(T, Rest), Sum is H + Rest.


sum_initial(X, X, []).
sum_initial([X | W], Z, [X | Y]):- sum_initial(W, Z, Y).

sum_perm(X, Y, Z):- permutation(X, Result), sum_initial(Result, Y, Z).

sum(X, Y, Z):- sum_perm(X, Y, Z), sum_list(Y, Sum1), sum_list(Z, Sum2), Sum1
=:= Sum2.
```

5.
```
mean(L,M ):-
        sumlist( L, Sum ),
        length( L, Length),
        (  Length > 0
        -> M is Sum / Length
        ;  M is 0
        ).
```

6.

a. listMovies(L):- findall(S, movie(S, _,_,_,_,_),L).
b. listMoviesByName(L):- listMovies(Result), sort(Result, L).
c. listMoviesByGenre(G, Result):- findall(L, (movie(L, _, Gs,_,_,_), member(G, Gs)), Result).
d. :- use_module(library(pairs)).

```
extract_negated_rank_and_title(movie(Title, _, _, _, _, Rank), NegRank-Title) :-
   NegRank is -Rank.

sort_movies_by_rank_desc(DB, SortedTitles) :-
   maplist(extract_negated_rank_and_title, DB, Pairs),
   keysort(Pairs, SortedPairs),
   pairs_values(SortedPairs, SortedTitles).

listMoviesByRank(DB, SortedTitles) :-
   sort_movies_by_rank_desc(DB, SortedTitles).
```

e. is_of_genre(genre(GenreName), movie(_, _, Genres, _, _, _)) :-

```
    member(genre(GenreName), Genres).

numberMovies(MoviesList, Genre, Count) :-
    include(is_of_genre(Genre), MoviesList, FilteredMovies),
    length(FilteredMovies, Count).
```