1. (a) \texttt{min_elem([Min],Min).}

\begin{verbatim}
\texttt{min_elem([Min|Tail], Min):-}
\texttt{min_elem(Tail,TailMin),}
\texttt{Min =< TailMin.}
\end{verbatim}

\begin{verbatim}
\texttt{min_elem([H|Tail],Min):-}
\texttt{min_elem(Tail,Min),}
\texttt{Min < H.}
\end{verbatim}

(b) query) Tries to prove \texttt{minElem([19,3,29], X)}.

1.1) Tries to prove first case \texttt{minElem([Min1], X)}.

1.1) Fails to match \texttt{[Min1]} to \texttt{[19,3,29]}.

2.1) Tries to prove second case \texttt{minElem([Head1|Tail1], X)}.

2.1) Matches \texttt{Head1} to \texttt{19} and \texttt{Tail1} to \texttt{[3,29]}.

2.2) Tries to prove \texttt{minElem([3,29], TailMin1)}.

2.2) Tries to prove second case \texttt{minElem([Head2|Tail2], TailMin1)}.

2.1) Matches \texttt{Head2} to \texttt{3} and \texttt{Tail2} to \texttt{[29]}.

2.2) Tries to prove \texttt{minElem([29], TailMin2)}.

1.1) Tries to prove first case \texttt{minElem([Min2], TailMin2)}.

1.1) Matches \texttt{[Min2]} to \texttt{[3,29]}.

1.1) Has proven \texttt{minElem([Min2], TailMin2)} for \texttt{TailMin2 = 29}.

2.2) Has proven \texttt{minElem([29], TailMin2)} for \texttt{TailMin2 = 29}.

2.3) Proves \texttt{Head2 \textless= TailMin2 \textless= 3} \textless= 29.

2.4) Proves \texttt{Min2 is Head2, for Min2 = 3}

2.1 Has proven \texttt{minElem([Head2|Tail2], TailMin1)} for \texttt{TailMin1 = 3}.

2.2) Has proven \texttt{minElem([3,29], TailMin1)} for \texttt{TailMin1 = 3}.

2.3) Fails to prove \texttt{Head1 \textless= TailMin1 \textless= 19 \textless= 3}.

3.1) Tries to prove third case \texttt{minElem([Head3|Tail3], X)}.

3.1) Matches \texttt{Head3} to \texttt{19} and \texttt{Tail3} to \texttt{[3,29]}.

3.2) Tries to prove \texttt{minElem([3,29], TailMin3)}.

1.1) Tries to prove first case \texttt{minElem([Min4], TailMin3)}.
1.1) Fails to match \([\text{Min}_4]\) to \([3,29]\).

2.1) Tries to prove second case \(\min\text{Elem}([\text{Head}_4|\text{Tail}_4], \text{TailMin}_3)\).

2.1) Matches \(\text{Head}_4\) to 3 and \(\text{Tail}_4\) to \([29]\).

2.2) Tries to prove \(\min\text{Elem}([29], \text{TailMin}_4)\).

1.1) Tries to prove first case \(\min\text{Elem}([\text{Min}_5], \text{TailMin}_4)\).

1.1) Matches \([\text{Min}_5]\) to \([29]\).

1.1) Has proven \(\min\text{Elem}([\text{Min}_5], \text{TailMin}_4)\) for \(\text{TailMin}_4 = 29\).

2.2) Has proven \(\min\text{Elem}([29], \text{TailMin}_4)\) for \(\text{TailMin}_4 = 29\).

2.3) Proves \((\text{Head}_4 = \lt \text{TailMin}_4) \iff (3 = \lt 29)\).

2.4) Proves \((\text{Min}_4 \text{ is Head}_4) \iff \text{Min}_4 = 3\).

2.1) Has proven \(\min\text{Elem}([\text{Head}_4|\text{Tail}_4], \text{TailMin}_3)\) for \(\text{TailMin}_3 = 3\).

3.2) Has proven \(\min\text{Elem}([3,29], \text{TailMin}_3)\) for \(\text{TailMin}_3 = 3\).

3.3) Proves \((\text{Head}_3 < \text{TailMin}_3) \iff (19 < 3)\).

3.4) Proves \((X \text{ is 3})\), for \(X \text{ = 3}\).

query) Has proven \(\min\text{Elem}([19,3,29], X)\) for \(X = 3\).

(c) \(\text{min\_elem([Min],Min)}:=-!\).

\[
\text{min\_elem([Min|Tail], Min)}:-
\text{min\_elem(Tail,TailMin),}
\text{Min =\lt TailMin,}!.
\]

\[
\text{min\_elem([_|Tail],Min)}:-
\text{min\_elem(Tail,Min)}.
\]

The first ! is required because the singleton list \([X]\) also matches with the pattern \([H|T]\), with \(H/X\) and \(T/[]\). Also the second ! is required because the two recursive definitions may both seem to match, but they are in fact mutually exclusive. Once ! is used, the third clause only matches if the second fails, and therefore we do not need to apply the test anymore.

2. \(\text{revlist([],[])}\).

\[
\text{revlist([H|T],RL)}:-
\text{revlist(T,RT),}
\text{append(RT,[H],RL)}.
\]

3. (a) \(\text{is\_a\_list([])}\).

\[
\text{is\_a\_list([_|_])}.
\]

\[
\text{elem(X)}:=- \text{\!+ is\_a\_list}(X).
\]

\[
\text{make\_flat([],[]}\).
\]

\[
\text{make\_flat([H|T],[H|Tflat]):-}
\text{elem(H),}!,
\text{make\_flat(T,Tflat)}.
\]

\[
\text{make\_flat([H|T],L)}:-
\]
make_flat(H, Hflat),
make_flat(T, Tflat),
append(Hflat, Tflat, L).

Notice the usage of ! to impose mutual exclusion between the different definitions of make_flat.

(b) Declaratively, the answers are all nested lists that, once flattened, give [a, b, c] as a result. Procedurally, by using the program above, Prolog returns:

\[
X = [a, b, c] \\
X = [a, b, c, []] \\
X = [a, b, c, [], []] \\
X = [a, b, c, [], [], []] \\
X = [a, b, c, [], [], [], []] \\
X = [a, b, c, [], [], [], [], []] \\
X = [a, b, c, [], [], [], [], [], []] \\
X = [a, b, c, [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []
\]

There are in fact infinitely many lists following this pattern, whose flattening results in [a, b, c].

4. equal([], []).
equal([H|T1], [H|T2]) :-
equal(T1, T2).

dom([H1], [H2]) :- H1 < H2.
dom([H|T1], [H|T2]) :-
\+ equal(T1, T2),
dom(T1,T2), !.
dom([H1|T1], [H2|T2]) :-
H1 < H2,
dom(T1,T2), !.
5. \% belongsTo(A,B,X) is true if X belongs to the interval \([A,B]\).

\begin{verbatim}
belongsTo(A, B, X) :-
    A < B,
    X is A.

belongsTo(A, B, X) :-
    A < B,
    A1 is A+1,
   belongsTo(A1, B, X).
\end{verbatim}

\% Every composite number has a factor less than or equal to its square root.
\% Therefore, it is not necessary to check beyond the square root.

\begin{verbatim}
notPrime(X) :-
    MaxDivisor is truncate(sqrt(X)),
   belongsTo(2,MaxDivisor,Divisor),
   M is X mod Divisor,
   M = 0.

isPrime(X) :- \+ notPrime(X).
\end{verbatim}

\begin{verbatim}
firstPrimeBetween(A,B,X) :-
   belongsTo(A,B,X),
isPrime(X),!.
\end{verbatim}

6. (i)

\% returns true if Divisor is the smallest factor of X

\begin{verbatim}
smallestFactor(Divisor,X) :-
    MaxDivisor is truncate(sqrt(X)),
   belongsTo(2,MaxDivisor,Divisor),
   M is X mod Divisor,
   M = 0,!.
\end{verbatim}

\% if there is no such Divisor up to the square root,
\% then X is a prime number and is itself the smallest divisor.

\begin{verbatim}
smallestFactor(X,X).
\end{verbatim}

\begin{verbatim}
factorization([],1) :- !.

factorization([H|T],X) :-
    smallestFactor(H,X),
    NewX is X div H,
    factorization(T,NewX).
\end{verbatim}
factorizationPower(FPow,X) :-
  factorization(F,X),
  transformPower(FPow,F).

transformPower([H,CPow|TPow], [H|T]) :-
  countSeqAndRemove([H|T], CPow, NewT),
  transformPower(TPow,NewT).

% transform lists like [2,2,2,3,3,5] from the factorization predicate
% into [2,3,3,2,5,1], both represent 2^3 * 3^2 * 5
transformPower([],[]).

countSeqAndRemove([H|T],H,Count,Y) :-
  countSeqAndRemove([H|T],H,Count,0,Y).

countSeqAndRemove([H|T],H2,FinalCount,CurrentCount,Y) :-
  NewCount is CurrentCount+1,
  countSeqAndRemove(T,H,FinalCount,NewCount,Y),!.

countSeqAndRemove([H|T],H2,FinalCount,CurrentCount,[H|T]) :-
  +(H=H2),
  FinalCount is CurrentCount.

countSeqAndRemove([],_,FinalCount,CurrentCount,[]) :-
  FinalCount is CurrentCount.

% Unicode encoding of the superscripts to represent exponents
superscript([8304],0).
superscript([185],1).
superscript([178],2).
superscript([179],3).
superscript([Y],X) :-
  X > 3,
  X < 10,
  Y is 8308 + X - 4.

% encoding for more than 1 digit exponents
superscript(Y,X) :-
  X >= 10,
  superscriptrev(Z,X),
  reverse(Z,Y).
superscriptrev([A|Y],X) :-
    X>10,
    Z is X mod 10,
    superscript([A],Z),
    NewX is X div 10,
    superscriptrev(Y,NewX),!.

superscriptrev(Y, X) :-
    superscript(Y,X).

% transforms the list representation into string representation
% with exponents
stringListOfFP([],[]).

% if the exponent is 1, then the transformed string consists of
% the factor and the multiplication symbol (Unicode 215)
stringListOfFP([L,215|LTail],[Factor,1|FTail]) :-
    number_codes(Factor,L), % L itself is a list,
    % which creates a list of list, which needs to be flattened later
    stringListOfFP(LTail,FTail).

% if the exponent is > 1, then the superscript of the exponent
% needs to be included.
stringListOfFP([L,E,215|LTail],[Factor,Power|FTail]) :-
    number_codes(Factor,L),
    superscript(E,Power),
    stringListOfFP(LTail,FTail).

printFactorization(S,X) :-
    factorizationPower(FPow,X),
    stringListOfFP(L,FPow),
    flatten(L,L1), % flattening the result from querying number_codes
    removeLast(L2,L1), % remove the last element
    % because there is always a dangling multiplication in the end
    name(S,L2),!.

removeLast(L1,L2) :-
    reverse(L2,[_|L3]),
    reverse(L3,L1).
7. perkm(1,1000,0.08).
   perkm(1001,10000,0.04).
   perkm(10001,20000,0.02).
   perkm(20001,99999999,0.0).

   euro(0,0).
   euro(Km, Price) :-
       perkm(Min,Max,P1),
       Km >= Min,
       Km =< Max,
       R is Min-1,
       euro(R, P2),
       Price is (Km-R)*P1+P2.