Problem 10; P1:

A)

```
valley(L)
valley_down(L)
valley_up(L)

valley_up([x1,x2|R]) = valley_up([x2|R]),       if x2>x1
                                 False,         otherwise

Valley_down([x1,x2|R]) = valley_down([x2|R]), if x2<x1
                                     False,                  otherwise


Valley(L) = false,                      if L < 3
                valley_down(L),         if first pair decreases
                false,                  if the first pair doesn't decrease
                valley_up(L),           if we find an increasing pair
                false,                  otherwise
```

B)

```
alt_sum(L,Sign,Acc)

alt_sum([x1|R],Sign,Acc) = Acc,                                    if L = []
                           alt_sum([R], -Sign, Acc + Sign * x1),   otherwise
```

Source code: A)

```
% valley(List)
% a. Check if a list has a 'valley'

valley([A,B|Rest]) :-
       A > B,                  % start decreasing
       valley_down([A,B|Rest]).

% first half: decreasing until we find a number bigger than the current number

valley_down([A,B|Rest]) :-
       ( B < A ->
              valley_down([B|Rest]);
       B > A ->
              valley_up([B|Rest])
       ).

% second half must be strictly increasing

valley_up([A,B|Rest]) :-
```

```
        B > A,
        valley_up([B|Rest]).

valley_up([_]).        % if it reached the end it succeded
```

B)

```
% alt_sum(List,Sum)
% b. Sum = alternating sum of the List

alt_sum(List, Sum) :-
    alt_sum(List, 1, 0, Sum).   % Sign is +1, accumulator is 0

% base case for empty list
alt_sum([], _, Acc, Acc).

% recursive case
alt_sum([A|Rest], Sign, Acc, Sum) :-
    NewAcc is Acc + A * Sign,
    NewSign is -Sign,
    alt_sum(Rest, NewSign, NewAcc, Sum).
```