Exercise 6.1

Let's call a list *doubled* if it is made of two consecutive blocks of elements that are exactly the same. For example, `[a,b,c,a,b,c]` is doubled (it's made up of `[a,b,c]` followed by `[a,b,c]`) and so is `[foo,gubble,foo,gubble]`. On the other hand, `[foo,gubble,foo]` is not doubled. Write a predicate `doubled(List)` which succeeds when `List` is a doubled list.

```
doubled(List) :- append(X,X,List).
```

Exercise 6.2

A palindrome is a word or phrase that spells the same forwards and backwards. For example, 'rotator', 'eve', and 'nurses run' are all palindromes. Write a predicate `palindrome(List)`, which checks whether List is a palindrome. For example, to the queries

```
?- palindrome([r,o,t,a,t,o,r]).
```

and

```
?- palindrome([n,u,r,s,e,s,r,u,n]).
```

Prolog should respond 'yes', but to the query

```
?- palindrome([n,o,t,h,i,s]).
```

Prolog should respond 'no'.

```
palindrome(List) :- reverse(List,List).
```

Exercise 6.3

1. Write a predicate `second(X,List)` which checks whether `X` is the second element of `List`.

```
second(X,[_,X|_]).
```

2. Write a predicate `swap12(List1,List2)` which checks whether `List1` is identical to `List2`, except that the first two elements are exchanged.

```
swap12([X,Y|T],[Y,X|T]).
```

3. Write a predicate `final(X,List)` which checks whether `X` is the last element of `List`.

```
final(X,List) :- reverse(List,[X|_]).
```

4. Write a predicate `toptail(InList,Outlist)` which says 'no' if ~~i~~`Inlist` is a list containing fewer than 2 elements, and which deletes the first and the last elements of `Inlist` and returns the result as `Outlist`, when `Inlist` is a list containing at least 2 elements. For example:

```
toptail([a],T).

no

toptail([a,b],T).

T=[]

toptail([a,b,c],T).

T=[b]
```

   Hint: here's where `append` comes in useful.

```
toptail([_|Xs],Outlist) :- append(Outlist,[_],Xs).
```

5. Write a predicate `swapfl(List1,List2)` which checks whether `List1` is identical to `List2`, except that the first and last elements are exchanged. Hint: here's where `append` comes in useful again.

```
swapfl([X|Xs],List2) :-
    append(T,[H],Xs),
    append([H|T],[X],List2).
```

Exercise 6.4

And here is an exercise for those of you who, like me, like logic puzzles.

There is a street with three neighboring houses that all have a different color. They are red, blue, and green. People of different nationalities live in the different houses and they all have a different pet. Here are some more facts about them:

- The Englishman lives in the red house.
- The jaguar is the pet of the Spanish family.
- The Japanese lives to the right of the snail keeper.
- The snail keeper lives to the left of the blue house.

- Who keeps the zebra?

Define a predicate `zebra/1` that tells you the nationality of the owner of the zebra.

Hint: Think of a representation for the houses and the street. Code the four constraints in Prolog. `member` and `sublist` might be useful predicates.

```
neighbor(L,R,[L,R|_]).

neighbor(L,R,[_|Xs]) :- neighbor(L,R,Xs).


zebra(X) :-

    Street = [H1,H2,H3],

    member(house(red,englishman,_), Street),

    member(house(_,spanish,jaguar), Street),

    neighbor(house(_,_,snail), house(_,japanese,_), Street),

    neighbor(house(_,_,snail), house(blue,_,_), Street),

    member(house(_,X,zebra),Street).
```