Practical 6

1. It is possible to write a one line definition of the `member` predicate by making use of `append`. Do so.

```
is_member(X,Y) :- append(_,[X|_],Y).
```

   How does this new version of `member` compare in efficiency with the standard one?

```
The append-version of member is slightly less efficient than the standard
member/2. While both are recursive and execute in linear time, there is an
extra step required to unify the variables used by append. The following
traces illustrate this.

?- member(3,[1,2,3]).

Call: (7) member(3, [1, 2, 3])

Call: (8) member(3, [2, 3])

Call: (9) member(3, [3])

Exit: (9) member(3, [3])

Exit: ...

?- is_member(3,[1,2,3]).

Call: (7) is_member(3, [1, 2, 3])

Call: (8) append(_L208, [3|_G381], [1, 2, 3])

Call: (9) append(_G384, [3|_G381], [2, 3])

Call: (10) append(_G387, [3|_G381], [3])

Exit: (10) append([], [3], [3])

Exit: ...
```

2. Write a predicate `set(InList,OutList)` which takes as input an arbitrary list, and returns a list in which each element of the input list appears only once. For example, the query

```
set([2,2,foo,1,foo, [],[]],X).
```

   should yield the result

```
X = [2,foo,1,[]]
```

   Hint: use the `member` predicate to test for repetitions of items you have already found.

```
set([],Acc,Acc).

set([H|Tail],Acc,OutList) :- member(H,Acc), set(Tail,Acc,OutList).

set([H|Tail],Acc,OutList) :- set(Tail,[H|Acc],OutList).

set([H|Tail],OutList) :- set(Tail,[H],OutList).
```

3. We 'flatten' a list by removing all the square brackets around any lists it contains as elements, and around any lists that its elements contain as element, and so on for all nested lists. For example, when we flatten the list

   `[a,b,[c,d],[[1,2]],foo]`

   we get the list

   `[a,b,c,d,1,2,foo]`

   and when we flatten the list

   `[a,b,[[[[[[c,d]]]]]],[[1,2]],foo,[]]`

   we also get

   `[a,b,c,d,1,2,foo]`

   Write a predicate `flatten(List,Flat)` that holds when the first argument `List` flattens to the second argument `Flat`. This exercise can be done without making use of `append`.

```
flatten([], Acc, Acc).

flatten(X, Acc, [X|Acc]) :- X\=[], X\=[_|_].

flatten([X|Xs], Acc, Result) :-
    flatten(Xs, Acc, NewAcc),
    flatten(X, NewAcc, Result).

flatten(List,Flat) :- flatten(List,[],Flat).
```