recursion

We have multiple ways to use recursion: in the database and in our queries for example we can make properties transitive.

```
dadof(you,dad).
dadof(dad,granddad).
dadof(granddad, petethegreat).
momof(petethegreat,eve)
ancestorof(X,Y):- dadof(X,Y).
ancestorof(X,Y):- momof(X,Y).
%naive: ancestorof(X,Y):- ancestorof(X,Z),ancestor(Z,Y).
```

recursion

We have multiple ways to use recursion: in the database and in our queries for example we can make properties transitive.

```
dadof(you,dad).
dadof(dad,granddad).
dadof(granddad, petethegreat).
momof(petethegreat,eve)
ancestorof(X,Y):- dadof(X,Y).
ancestorof(X,Y):- momof(X,Y).
%naive: ancestorof(X,Y):- ancestorof(X,Z),ancestor(Z,Y).

%better:
ancestorof(X,Y):- dadof(X,Z),ancestor(Z,Y).
ancestorof(X,Y):- momof(X,Z),ancestor(Z,Y).
```

recursion on lists

suppose we have a list of groupmembers `[roald, winand, anvar, alexey, sjoerd]`

recursion on lists

suppose we have a list of groupmembers `[roald, winand, anvar, alexey, sjoerd]`
we can declare a member of this group to be part of this list using recursion i.e.

```
member(H, [H|T])

member(X, [H|T]) :- member(X,T)
```

recursion on lists

suppose we have a list of groupmembers `[roald, winand, anvar, alexey, sjoerd]`
we can declare a member of this group to be part of this list using recursion i.e.

```
member(H, [H|T])

member(X, [H|T]) :- member(X,T)
```

So when we search for members we can simply search
`?- member(anvar,[roald, winand, anvar, alexey, sjoerd])`
which would say yes

recursion

in prolog we do have to be careful when using recursion, for example with:

```
something :- something.
```

...
we also want to use tail recursion in favour of head recursion:

```
lengthH([],0).
lengthH([H|T],N):- lengthH(T,X), N is X+1.
lengthT([H|T],A,L) :- Anew is A+1, lengthT(T,Anew,L).
lengthT([],L,L).
lengthTail(List,L) :- lengthT(List,0,L). %to make it nicer
```