

Doing business with lists

- Lists look like this:
[1, 2, 3, 4] or [a, b, c] or [1, a, b]
- Empty lists are written as [].
- The notation [H|T] refers to the *head* H of a list and its *tail* T. Thus, if [H|T] *unifies* (make sure you know what this means!) with [1, 2, 3], then H= 1 and T=[2, 3]. If it unifies with [1], then H=1 and T=[].
- [a,b,c] doesn't unify with [b|T]
- [] doesn't unify with [H|T]
- [] unifies with []

List processing

- Load a file consisting of the following fact: `p([H|T], H, T)`.
- Run the following queries:
 - `?- p([a,b,c], X, Y)`.
 - `?- p([a], X, Y)`.
 - `?- p([], X, Y)`.
- Define a predicate `member(Element, List)` such that it returns true if `Element` is contained in `List`. What happens if you pose this query with an uninstantiated variable `Element` and an instantiated `List`? What happens if `Element` is instantiated and `List` is not? (Code on the next slide).

More list processing

- The code for `member`:

```
member(X,[X|R]).  
member(X,[Y|R]) :- member(X,R).
```
- Run the following queries as well:
 - `?- member([3,Y], [[1,a],[2,m],[3,z],[4,v],[3,p]]).`
 - `?- member(X,[23,45,67,12,222,19,9,6]), Y is X*X, Y < 100.`
- Now write code for a predicate that takes a list and outputs a list consisting of those elements of the first list which are greater than, say, 20. (This should be easy).

Still more fun with lists

- Code for appending two lists:

```
append([X|Y],Z,[X|W]) :- append(Y,Z,W).  
append([],X,X).
```

- Write code for a predicate `remove(Element, List1, List2)` which removes all instances of `Element` from `List1` to produce `List2`.