Using `select`

```
?- select(2,[1,2,3],Z).

Z = [1, 3] ;

No
?- select(2,Y,[1,3]).

Y = [2, 1, 3] ;

Y = [1, 2, 3] ;

Y = [1, 3, 2] ;

No
```

The `reverse` Predicate

```
?- reverse([1,2,3,4],Y).

Y = [4, 3, 2, 1] ;

No
```

- Predefined **`reverse(X,Y)`** unifies **`Y`** with the reverse of the list **`X`**

An Implementation

```
reverse([],[]).
reverse([Head|Tail],X) :-
   reverse(Tail,Y),
   append(Y,[Head],X).
```

- Not an efficient way to reverse!
- Later we'll see why, and a more efficient solution

When Queries Go Bad

```
?- reverse(X,[1,2,3,4]).

X = [4, 3, 2, 1] ;

Action (h for help) ? a
% Execution Aborted
?-
```

- Asking for another solution caused an infinite loop
- Hit control–C to stop it, then **a** for abort
- **reverse** cannot be used as flexibly as **append**

Flexible and Inflexible

- Ideally, predicates should all be flexible like **`append`**
- They are more declarative, with fewer procedural quirks to consider
- But inflexible implementations are sometimes used, for efficiency or simplicity
- Another example is **`sort`**…

Example

```
?- sort([2,3,1,4],X).

X = [1, 2, 3, 4] ;

No
?- sort(X,[1,2,3,4]).
ERROR: Arguments are not sufficiently instantiated
```

- A fully flexible **`sort`** would also be able to unsort—find all permutations
- But it would not be as efficient for the more common task

The Anonymous Variable

- The variable _ is an anonymous variable

- Every occurrence is bound independently of every other occurrence

Example

```
tailof(.(_,A),A).
```

- This **`tailof(X,Y)`** succeeds when **`X`** is a non-empty list and **`Y`** is the tail of that list
- Don't use this, even though it works:

```
tailof(.(Head,A),A).
```

Dire Warning

```
append([], B, B).
append([Head|TailA], B, [Head|TailC]) :-
  append(TailA, B, Tailc).
```

- Don't ignore warning message about singleton variables

- *If you misspell a variable name, this is the only warning you will see*