Prolog Examples

Example - 1

◊ Write a goal to delete the last three elements from a list, L, to produce another list, L1.

Example - 1

◊ Write a goal to delete the last three elements from a list, L, to produce another list, L1.

append(L1, [_, _, _], L).

Example - 2

◊ Write a goal to delete the first three elements and the last three elements from a list, L, to produce another list, L2.

Example - 2

◊ Write a goal to delete the first three elements and the last three elements from a list, L, to produce another list, L2.

```
append( [ _, _, _ | L2 ],  [ _, _, _], L ).
```


Example - 3

◊ Define the predicate last(Item, List) which asserts that Item is the last element of a list, List.

◊ Write 2 versions

» **Using append**

» **Without using append**

Example - 3

◊ Define the predicate last(Item, List) which asserts that Item is the last element of a list, List.

◊ Write 2 versions

» **Using append**

```
last( Item, List ) :-
   append( _, [Item], List).
```

» **Without using append**

Example - 3

◊ Define the predicate last(Item, List) which asserts that Item is the last element of a list, List.

◊ Write 2 versions

» **Using append**

```
last( Item, List ) :-
   append( _, [Item], List).
```

» **Without using append**

```
last( Item, [Item] ).
last( Item, [ Head | Tail ] ) :-
   last( Item, Tail ).
```

Example - 4

- Define the predicate shift(List1, List2) which asserts that List2 is List1 shifted rotationally by one-element to the left.
- For example:

```
?- shift( [ 1, 2, 3, 4, 5 ], L1 ), shift( L1, L2 ).
   L1 = [ 2, 3, 4, 5, 1 ]
   L2 = [ 3, 4, 5, 1, 2 ]
```


Example - 4

- Define the predicate shift(List1, List2) which asserts that List2 is List1 shifted rotationally by one-element to the left.
- For example:

```
?- shift( [ 1, 2, 3, 4, 5 ], L1 ), shift( L1, L2 ).
   L1 = [ 2, 3, 4, 5, 1 ]
   L2 = [ 3, 4, 5, 1, 2 ]

   shift( [ Head | Tail ], Shifted ) :-
     append( Tail, [ Head ], Shifted ).
```


Example - 5

- Define two predicates evenlength(List) and oddlength(List) that assert that their arguments are lists of even or odd length respectively.

Example - 5

- Define two predicates evenlength(List) and oddlength(List) that assert that their arguments are lists of even or odd length respectively.

```
evenlength( [ ] ).
evenlength( [ Head | Tail ] ):-
   oddlength( Tail ).

oddlength( [ _ ] ).
oddlength( [ Head | Tail ] ) :-
   evenlength( Tail ).
```

Example - 6

◊ Define the predicate reverse(List, ReversedList) that asserts that ReversedList is a list whose elements are in the opposite order to List.

Example - 6

◊ Define the predicate reverse(List, ReversedList) that asserts that ReversedList is a list whose elements are in the opposite order to List.

```
reverse( [ ], [ ] ).
reverse( [ Head | Tail ], Reversed ) :-
   reverse( Tail, ReversedTail ),
   append( ReversedTail, [ Head ], Reversed ).
```


Example - 7

◊ Define the predicate reverse(List, ReversedList) that asserts that ReversedList is a list whose elements are in the opposite order to List. Use an accumulator.

Example - 7

◊ Define the predicate reverse(List, ReversedList) that asserts that ReversedList is a list whose elements are in the opposite order to List. Use an accumulator.

```
reverse( List, Result ) :- reverse( List, [ ], Result ).

reverse( [ ], Result, Result ).
reverse( [ Head | Tail ],  SoFar, Result ) :-
   reverse( Tail, [ Head | SoFar ], Result ).
```

Example - 8

◊ Define a predicate sumlist(List, Sum) that asserts that List is a list of numbers and Sum is their sum. Use an accumulator.

Example - 8

◊ Define a predicate sumlist(List, Sum) that asserts that List is a list of numbers and Sum is their sum. Use an accumulator.

```
sumlist( List, Sum ) :-
   sumlist( List, 0, Sum ).

sumlist( [ ], Sum, Sum ).
sumlist( [ Head | Tail ], Partial, Total ) :-
   NewPartial is Partial + Head,
   sumlist( Tail, NewPartial, Total ).
```