# CT331 Assignment 3

## Question 1

Given the following facts

```
takes(tom, ct331).
takes(mary, ct331).
takes(joe, ct331).
takes(tom, ct345).
takes(mary, ct345).
instructs(bob, ct331).
instructs(ann, ct345).
```

### Write a prolog rule called ‘teaches’ that returns true if a given instructor teaches a given student

```
teaches(Instructor, Student) :-
    instructs(Instructor,Course),
    takes(Student, Course).
```

### Write a prolog query that uses the ‘teaches’ rule to show all students instructed by bob

```
teaches(bob, Student).
```

## Write a prolog query that uses the ‘teaches’ rule to show all instructors that instruct mary

```
teaches(Instructor, mary).
```

## What is the result of the query: teaches(ann, joe). Why is this the case?

The result of

```
teaches(ann, joe).
```

is false, as prolog could not find an instructor called ‘ann’ that teaches joe in its database. There is no explicit relationship between ‘ann’ and ‘joe’.

## Write a prolog rule called ‘classmates’ that returns true if two students take the same course

```
classmates(Student1, Student2) :-
  takes(Student1, Course),
  takes(Student2, Course).
```

Example queries

```
classmates(tom, mary).
% tom and mary both share ct331, and ct345
% returns
true;
true.

classmates(joe, mary).
% joe and mary share ct331 but not ct345
% returns
true;
false.
```

## Question 2

### Using the “=” sign and the prolog list syntax to explicitly unify variables with parts of a list, write a prolog query that displays the head and tail of the list [1, 2, 3]

```
[H | T] = [1, 2, 3].
% returns
H = 1
T = [2, 3]
```

### Use a nested list to display the head of the list, the head of the tail of the list and the tail of the tail of the list [1 , 2, 3, 4, 5]

```
[H1, H2 | T ] = [1,2,3,4,5].
% returns
H1 = 1
```

```
H2 = 2
T = [3, 4, 5]
```

## Write a prolog rule ‘contains1’ that returns true if a given element is the first element of a given list

```
contains1(Lst, El) :-
    [El | _] = Lst.

% result
contains1([1,2,3], 2).
false.

contains1([1,2,3], 1).
true.
```

## Write a prolog rule ‘contains2’ that returns true if a given list is the same as the tail of another given list

```
contains2(Lst1, Lst2) :-
    [ _ | Lst1 ] = Lst2.

% result
contains2([2,3], [1,2,3]).
true.

contains2([1,2], [1,2,3]).
false.
```

## Write a prolog query using ‘contains1’ to display the first element of a given list

```
contains1([1,2,3,4], X).
% returns
1.
```

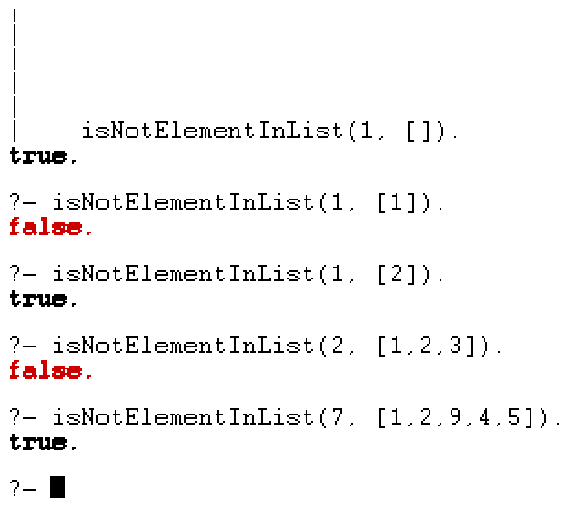
# Question 3

## Write a set of prolog facts and rules called isNotElementInList that determine if a given element is not in a given list. The element must be the first argument and the list must be the second

```
% base case, list is empty, element not in list
isNotElementInList(_, []) :-
    !, true.

% element matches the head
isNotElementInList(El, [El | _]) :-
    !, false.

% Doesn't match head, recurse through tail
isNotElementInList(El, [_ | T]) :-
    isNotElementInList(El, T).
```



```
|
|
|
|
|
|    isNotElementInList(1, []).
true.

?- isNotElementInList(1, [1]).
false.

?- isNotElementInList(1, [2]).
true.

?- isNotElementInList(2, [1,2,3]).
false.

?- isNotElementInList(7, [1,2,9,4,5]).
true.

?- 
```

## Question 4

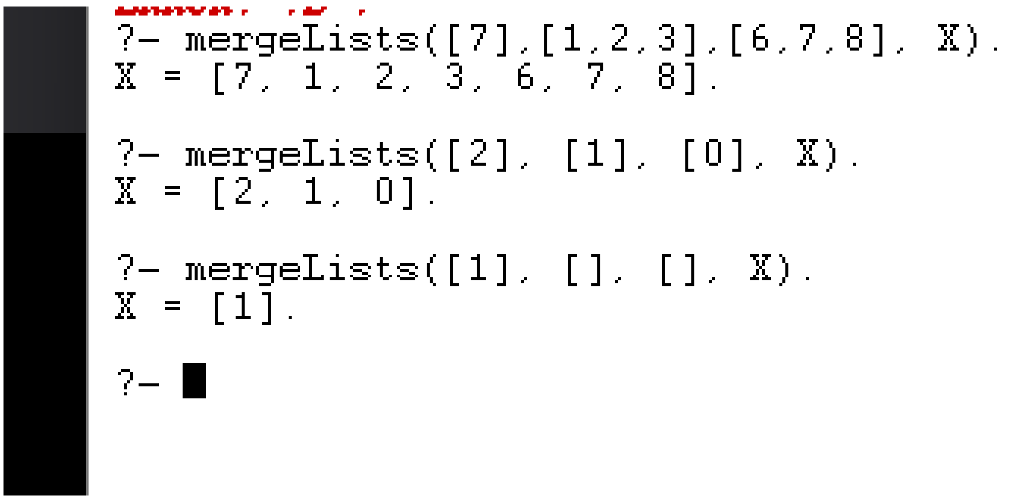
**Write a set of prolog facts and rules called mergeLists that merges (concatenates / appends) three lists. The given lists must be the first three arguments, and the merged list must be the fourth argument**

```
% base case, combine remaining lists
mergeLists([], List, List).

% Add head of list to merged, and recurse through tail
mergeLists([H|T], List3, [H|Merged]):-mergeLists(T, List3, Merged).

% If first list is now empty, move on to list2
mergeLists([], List2, List3, Merged):- mergeLists(List2, List3, Merged).

% Add head of list to merged, and recurse through tail
mergeLists([H|T], List2, List3, [H|Merged]):- mergeLists(T, List2, List3, Merged).
```



```
?- mergeLists([7],[1,2,3],[6,7,8], X).
X = [7, 1, 2, 3, 6, 7, 8].

?- mergeLists([2], [1], [0], X).
X = [2, 1, 0].

?- mergeLists([1], [], [], X).
X = [1].

?-
```

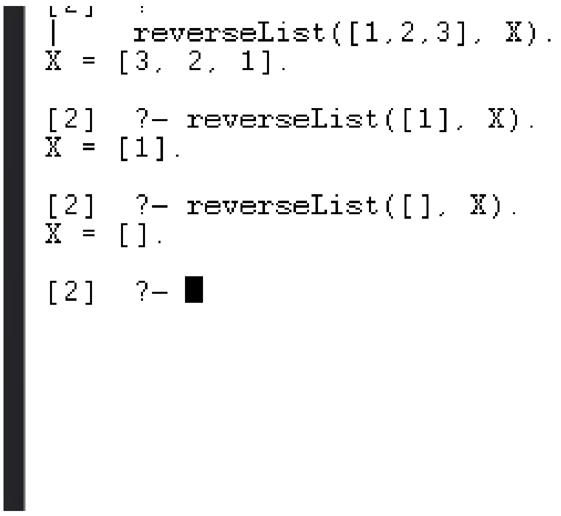
## Question 5

### Write a set of prolog facts and rules called reverseList that reverses a given list. The given list must be the first argument and the reversed list must be the second

```
% Caller function
reverseList(List, Reversed) :- reverseList(List, [], Reversed).

% Base case, combine head and reversed list, when tail is empty
reverseList([], Rev, Rev).

% Insert head into the middle
reverseList([H|T], X, Reversed) :-
    reverseList(T, [H|X], Reversed).
```



```
|    reverseList([1,2,3], X).
X = [3, 2, 1].

[2]  ?- reverseList([1], X).
X = [1].

[2]  ?- reverseList([], X).
X = [].

[2]  ?- 
```

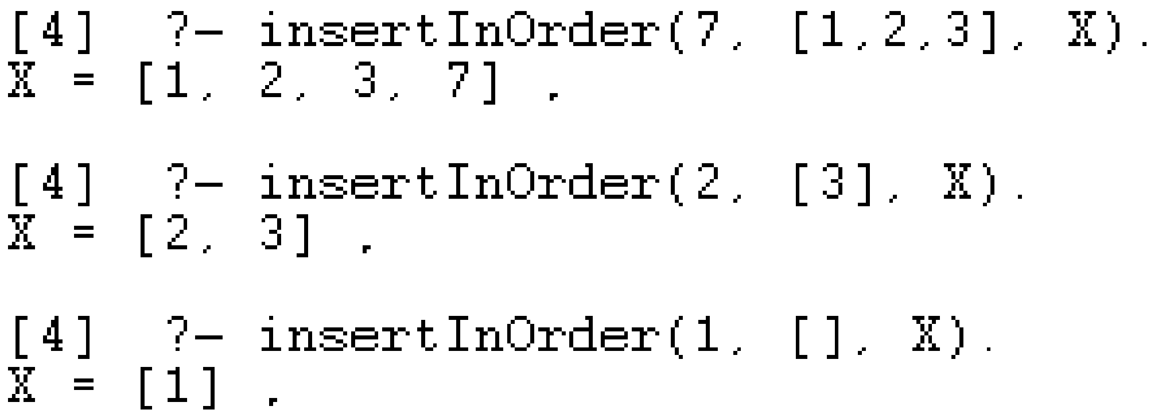
## Question 6

**Write a set of prolog facts and rules called insertInOrder that inserts an element into its correct position in a given list. The given element must be the first argument, the given list must be the second and the final list with the inserted element must be the third argument. You can assume that the 2nd argument is sorted in ascending orde**

```
% base case, El is the biggest number.
insertInOrder(El, [], [El]).

% El is less then Head, hence insert before
insertInOrder(El, [H | T], [El, H | T]) :-
    El < H.

% El, is bigger than H, recurse through list
insertInOrder(El, [H | T], [H | NewList]) :-
    El > H,
    insertInOrder(El, T, NewList).
```



```
[4]  ?- insertInOrder(7, [1,2,3], X).
X = [1, 2, 3, 7] .

[4]  ?- insertInOrder(2, [3], X).
X = [2, 3] .

[4]  ?- insertInOrder(1, [], X).
X = [1] .
```