

Question 1.

```
/*
1. List of possible suspects. Note that names are entered in lower case,
   as they are all constants.
*/
possible_suspect(fred).
possible_suspect(mary).
possible_suspect(jane).
possible_suspect(george).

/*
2. The crime log. Once again, all constants are entered in lower case
*/
crime(robbery1, john, tuesday, park).
crime(assault2, robin, thursday, park).
crime(robbery2, jim, wednesday, pub).
crime(assault1, mary, wednesday, park).

/*
3. Facts on the whereabouts of suspects on given days
*/

/* Fred */
was_at(fred, tuesday, park).
was_at(fred, wednesday, pub).
was_at(fred, thursday, pub).

/* George */
was_at(george, tuesday, pub).
was_at(george, wednesday, pub).
was_at(george, thursday, home).

/* Jane */
was_at(jane, tuesday, home).
was_at(jane, wednesday, park).
was_at(jane, thursday, park).

/* Mary */
was_at(mary, tuesday, pub).
was_at(mary, wednesday, park).
was_at(mary, thursday, home).

/*
4. Who is jealous of whom,
*/
jealous_of(fred, john).
jealous_of(jane, mary).

/*
   and who owes money to whom.
*/
owes_money_to(george, jim).
owes_money_to(mary, robin).
```

```
/*
5. Note the use of capital letters to denote variables in the rules below.
*/
```

```
/* A Person is a prime suspect in a Crime, if the Crime took place against
a Victim at a Time and Place, the Person is a possible_suspect, the Person
was_at the Time and Place, and the Person had_motive_against the Victim. */
```

```
prime_suspect(Person, Crime) :-
    crime(Crime, Victim, Time, Place),
    possible_suspect(Person),
    was_at(Person, Time, Place),
    had_motive_against(Person, Victim).
```

```
/* We now define the predicate had_motive_against by two rules. */
```

```
/* Person had_motive_against Victim if Person was jealous_of Victim, */
had_motive_against(Person, Victim) :-
    jealous_of(Person, Victim).
```

```
/* or Person had_motive_against Victim if Person owes_money_to Victim. */
had_motive_against(Person, Victim) :-
    owes_money_to(Person, Victim).
```

```
/*
6.
?- prime_suspect(fred, robbery1).
yes.
```

```
?- prime_suspect(george, assault1).
no
```

```
?- prime_suspect(Who, robbery1).
Who = fred;
no
```

```
?- prime_suspect(Who, robbery2).
Who=george;
no
```

```
?- prime_suspect(Who, assault1).
Who=jane;
no.
```

```
?- prime_suspect(Who, assault2).
no.
```

```
?- prime_suspect(fred, What).
What = robbery1;
no.
```

```
?- prime_suspect(george, What).
What = robbery2;
no.
*/
```

Question 2.

```
/*
For the first three questions there are always two immediate solutions which
come to my mind, the first one uses append, the second is by recursion.
```

```
1. X is the last element of list L, if  $L = K @ [X]$  for some list K
```

```
*/
last(X,L) :- append(_, [X], L).
```

```
/* The recursive solution:
```

```
last(X, [X]).
last(X, [_|T]) :- last(X, T).
```

```
2. The reverse of  $[X|K]$  can be written as (the reverse of K) @ [X] .
```

```
*/
reverse([], []).
reverse([X|K], L) :- reverse(K, M), append(M, [X], L).
```

```
/* Again there is also a (more efficient) recursive solution:
```

```
reverse(L, RL) :- reverse(L, [], RL).
```

```
reverse([], RL, RL).
```

```
reverse([H|T], RPL, RL) :- reverse(T, [H|RPL], RL).
```

```
3. L is the result of removing exactly one occurrence of X from K,
iff  $K = A @ [X] @ B$  and  $L = A @ B$  for some A, B.
```

```
*/
select(X, K, L) :- append(A, [X|B], K), append(A, B, L).
```

```
/* and a recursive solution:
```

```
select(X, [X|L], L).
select(X, [Y|K], [Y|L]) :- select(X, K, L).
```

```
4. If L is a list without repetitions, then
X and Y are two different members of L, if after removing one occurrence
of X from L, Y still occurs in the resulting list.
```

```
*/
different(X, Y, L) :- select(X, L, M), member(Y, M).
```

```
/*
```

```
5. A solution for samelength which never loops.
```

```
*/
samelength([], []).
samelength([_|K], [_|L]) :- samelength(K, L).
```

```
/*
```

```
6. Two lists  $[X|K]$  and L of the same length are similar if
K is similar to the result of removing one occurrence of X from L.
```

```
*/
similar2([], []).
similar2([X|K], L) :- select(X, L, M), similar2(K, M).
```

```
similar(K, L) :- samelength(K, L), similar2(K, L).
```