Problem 1. For this question we will be modifying and querying the file *dg.pl*.

1. Write a query to return all children in the knowledge base.
2. Add a predicate *siblings/2* to the knowledge base which holds for all pairs of siblings. Two people are siblings if they share at least one parent.
3. Add a similar predicate *cousins/2* which holds for all cousins (i.e. All pairs of people for whom at least one of their parents is siblings with at least one parent of the other). Also add using 2 explicit facts that michelle and james are cousins.
4. Add a predicate *friends_sym/2* implementing a version of *friends/2* which is symmetric. Why can't we simply add a rule:

   ```
   friends(X,Y) :- friends(Y,X).
   ```

5. Suppose we wanted to query all people with cousins but no friends. Does the query `friendless_cousins(X)` give the same result under both of the the following definitions of `friendless_cousins/2`? Why or why not?

   ```
   friendless_cousins(X) :- cousins_sym(X,Z), \+ friends(Y,X).
   ```

   ```
   friendless_cousins(X) :- \+ friends(Y,X), cousins_sym(Z,X).
   ```

Problem 2. Like Haskell, Prolog has cons lists, with X cons Y written as `[X|Y]`. For syntactic sugar, we can write lists in the usual way `[1,2,3]`. The empty list is `[]`.

Consider the predicate `member/2` which is determines whether an element is a member of a list. It can be defined recursively as:

```
member(X, [X|T]).
member(X,[Y|T]) :- member(X,T).
```

Write your own version of the predicate `remove/3`. `remove(X, L1, L2)` is satisfied if L2 is the result of removing the first occurrence of X from L1. It should fail if X is not in L1.

Why do we not need a base case for these recursions?

Problem 3. Write a predicate `maximum/2` which you can use to determine the maximum element of a list of integers.

Problem 4. One way of representing a directed graph in Prolog is using an predicate `adj/2` to denote adjacent edges. Write a predicate `path/2` which is satisfied if there is a path between two edges.

Consider a graph with cycles. Ensure that your predicate `path/2` cannot get stuck in a cycle. Write a predicate `cycle/1` to determine if a node is in a cycle.