Logical reasoning and programming, **lab session 10**
(November 21, 2022)

In this lab, we start working with Prolog. Prolog is a logic programming language. It is (mostly) a declarative programming language used to write logic programs. The Prolog runtime then attempts to find answers to our queries given the program.

We will be using `SWI-Prolog` implementation of Prolog. You may download it from their website. Most Linux distributions should also have it available in their repositories. Last but not least, people behind `SWI-Prolog` offer an online sandbox environment SWISH.

Task 1: Familiarize yourself with the House of Windsor, the reigning royal house of the United Kingdom. We will use a part of their family tree as the ground facts of our programs. Consider the following people:

- `george`: George VI, King of the UK and the British Dominions
- `elizabeth`: Elizabeth II, Her Majesty Queen Elizabeth
- `philip`: Prince Philip, Duke of Edinburgh
- `margaret`: Princess Margaret, Countess of Snowdon
- `charles`: Charles III, HM King, Head of the Commonwealth
- `diana`: Diana, Princess of Wales
- `camilla`: Camilla, Queen Consort
- `edward`: Prince Edward, Earl of Wessex and Forfar
- `sophie`: Sophie, Countess of Wessex and Forfar
- `william`: William, Prince of Wales
- `kate`: Catherine, Princess of Wales
- `harry`: Prince Harry, Duke of Sussex
- `meghan`: Meghan, Duchess of Sussex
- `louise`: Lady Louise Windsor
- `james`: James, Viscount Severn

and their relationships:

- `male(X)` means that `X` is a male.
- `female(X)` if `X` is a female.
- `parent(P,C)` if `P` is the parent of `C`. Not the other way around!
- `wife(W,H)` if `W` is (or was) the wife of `H`.

```
female(elizabeth).
female(margaret).
female(diana).
female(camilla).
female(sophie).
female(kate).
female(meghan).
female(louise).

male(george).
male(philip).
male(charles).
male(edward).
male(william).
male(harry).
male(james).

parent(george,elizabeth).
parent(george, margaret).
parent(elizabeth,charles).
parent(philip,charles).
parent(elizabeth,edward).
parent(philip,edward).
parent(charles,william).
parent(diana,william).
parent(charles,harry).
parent(diana,harry).
parent(edward,louise).
parent(sophie,louise).
parent(edward,james).
parent(sophie,james).

wife(elizabeth,philip).
wife(diana,charles).
wife(camilla,charles).
wife(sophie,edward).
wife(kate, william).
wife(meghan, harry).
```

Copy&Paste the code above into a file "`royal.pl`". It is loaded into Prolog by the "`[filename]`" command. The console should look like:

```
?- [royal].
true.
```

If you are using SWISH, the program is loaded automatically. No need for this step, then.

Task 2: Write a query to ask for all children of `elizabeth`.

Task 3: Define the predicate `husband(Man,Woman)`.
Do not list all husbands of all wives as ground facts! :-)

Task 4: Define `person(P)` to be either a male or a female.
Try avoiding the `;` symbol (which defines a logical *or*).

Task 5: Define `mother(Mother,Child)` and `father(Father,Child)`.
Be careful not to define a son or a daughter.

Check your knowledge:

- What is a difference between a `Person` and a `person` ?
- What is a *singleton*? Should you avoid it? How does an underscore `_` fit into it?

Task 6: Compare the two definitions of the `father` predicate:
a) `father1(F,C) :- male(F), parent(F,C).`
b) `father2(F,C) :- parent(F,C), male(F).`
Which of them is faster on the `father`*`[1/2]`*`(X,charles)` query? Why?

Note: You can estimate Prolog's speed by starting the *trace* mode:

```
?- trace.
true.
[trace] ?-
```

It can be turned off by the `nodebug` command.

Task 7: Write two SLD trees for `father1(X, charles)` and `father2(X, charles)`.

Check your knowledge:

- What is a *left-to-right* rule?
- What is a *top-to-bottom* rule?

Task 8: Define the `ancestor(Ancestor,Descendant)`, which connects the `Ancestor` with any of their descendants.

Task 9 (optional): Define the `sibling(A,B,Parent)` predicate such that `A` is the sibling of `B` and `Parent` is their shared parent. Be careful, no person is its own sibling. Therefore `sibling(william,william,P)` should fail!
Hint: Use the non-unifiability predicate "`\=`".
Note: "`\=`" is in fact just syntactic sugar for applying a technique known as ***negation as failure*** *to the Prolog unification procedure. We will study negation as failure later on in the course. If you are interested, you may use the course literature (or Google) to read up on it in advance.*