AIT Semantic and Declarative Technologies Course
Class Practice P2-1: First steps in Prolog programming

For each task write a Prolog predicate that corresponds to the specification provided as a comment. You can use built-in predicates mentioned in the lecture slides so far, e.g. `X = Y` and `Y is Expr`.
The following input-output mode indicators are used in the specs to annotate the arguments:

- `+` – input argument, i.e., it cannot be an unbound variable;
- `-` – output argument, i.e., it has to be an unbound variable;
- `?` – the argument might be either input or output.

A sample task with solution

Given a list `L` and an integer `N`, check that the length of `L` is `N`.

```
% list_length(+L, +N): List L has N elements.

| ?- list_length([], 0).                ⟹          yes
| ?- list_length([], 3).                ⟹          no
| ?- list_length([a,b,c,d], 4).         ⟹          yes
| ?- list_length([a,b,c,d], 2).         ⟹          no
```

A solution:

```
% list_length(+L, +N): List L has N elements
list_length([], R) :-
        R = 0.
list_length([_Head|Tail], R):-          % See note below re _Head
        R1 is R-1,
        list_length(Tail, R1).
```

Note that using the underline as the first character of the variable name `_Head` we tell the Prolog compiler that we are aware that the value of this variable is thrown away, i.e. that it is of no interest to us. If we omit the underline character, the code will still be working, but we get a "singleton variable" warning (a variable occurs only once in the clause).

Class practice tasks

1. Write a predicate that checks that a list is nonempty and all its elements are the same. Let's call such a list A-boring, where A is the element appearing repeatedly.

```
% boring(+L, ?A): List L is A-boring.

| ?- boring([2,2], A).                  ⟹          A = 2 ? ; no
| ?- boring([2,2.1], A).                ⟹          no
| ?- boring([a,a,a], a).                ⟹          yes
| ?- boring([1+2,2+1], A).              ⟹          no
| ?- L = [A,2], boring(L, X).           ⟹          L = [2,2], A = 2, X = 2 ? ; no          (*)
| ?- length(L, 4), boring(L, X).        ⟹          L = [X,X,X,X] ? ; no                    (*)
```

2. Given a list of numbers, calculate the sum of the list elements. (L is a list of numbers.)

```
% sum(+L, ?Sum): L sums up to Sum.

| ?- sum([], S).                        ⟹          S = 0 ? ; no
| ?- sum([0.5,7,1.5], S).               ⟹          S = 9.0 ? ; no
| ?- sum([2,3,1], 6).                   ⟹          yes                                     (*)
| ?- sum([2,3], 6).                     ⟹          no                                      (*)
```

3. Given two arbitrary lists, check that they are of equal length.

```
% same_length(+L1, ?L2): Lists L1 and L2 are of equal length.

| ?- same_length([1,2], [a,b]).         ⟹          yes
| ?- same_length([1,2], [a]).           ⟹          no
| ?- same_length([1,2], L2).            ⟹          L2 = [_A,_B] ? ; no                     (*)
| ?- same_length([1,2], L2), boring(L2, a). ⟹      L2 = [a,a] ? ; no                       (*)
```

Important remark: When writing your program, you need not worry about test cases marked with (*). In general, these should work "automatically", when the previous ones are OK.