Linguagens de Programação: Lab Session 1

The goal of this lab session is to continue to explore the Coq proof assistant.

Important: Before you tackle the exercises listed below, please make sure that you have covered the material discussed in Lecture 1.

Exercise 1

In Lecture 1, we defined the following datatype:

```
Inductive day : Type :=
  | monday : day
  | tuesday : day
  | wednesday : day
  | thursday : day
  | friday : day
  | saturday : day
  | sunday : day.
```

1.1. Define a new function, called `weekday_to_nat`, that maps week days to numbers from `1` to `7` (where `sunday` is `1`, `monday` is `2`, ..., and `saturday` is `7`).

1.2. Test your definition by applying your function to the different days.

1.3. Define the function `is_weekend` that, given a day `d`, returns `true` if `d` is a weekend day and `false` otherwise. Use pattern matching and the *wildcard pattern*.

Exercise 2:

In Lecture 1, we also discussed booleans, with members `true` and `false`.

```
Inductive bool : Type :=
  | true : bool
  | false : bool.
```

2.1. Define the functions `negb`, `andb`, and `orb` as shown in Lecture 1.

2.2. We saw that we can write "unit tests" as assertions of the form:

```
Example test_orb1: (orb true false) =
```

```
true.
```

To prove assertions, we can also write proofs like the following:

```
Proof. simpl. reflexivity. Qed.
```

Following the same approach, write the three other possibilities for `orb` that need to be tested and prove them.

2.3. Introduce the following new notations and experiment with them.

```
Notation "~ x" := (negb x).
Notation "x && y" := (andb x y).
Notation "x || y" := (orb x y).
```

2.4. Define the function `xor` that corresponds to the *Exclusive Or*. Using `Example`, write assertions for each possible case and prove them.

Exercise 3

Note: the definitions asked below in 3.1 and 3.2 should be done in a new module called `NatPlayground`. Start by creating the new module using the keyword `Module` (as shown in the lectures).

3.1. Define your own type of natural numbers.

3.2. Define the function `minustwo` that we discussed in Lecture 1.

Note: if you want to test this function using arabic numerals (e.g. 1, 4, 42, etc.), comment out your own `nat` definition.

3.3. Define recursively the function `evenb` that, given a natural number `n`, returns `true` if the number is even and returns false otherwise.

3.3. Define recursively the function `oddb` that, given a natural number `n`, returns `true` is the number is odd and returns false otherwise.

3.4. Define `oddb` in terms of `evenb` and `negb`.

3.5. Define recursively the functions `plus`, `mult`, and `exp`, that correspond to addition, multiplication, and exponentiation (respectively). Test your definitions.

3.6. Write down the simplification steps that Coq performs to evaluate the value of `plus 3 2`.

3.7 Define **recursively** the function `minus` that corresponds to subtraction on natural numbers. Note that the smallest natural number is `0`, so whenever the first argument of `minus` is smaller or equal to the second, the function should return `0`.

3.8. Define the `factorial` function. Recall that its mathematical definition is:

```
factorial(0)  =  1
factorial(n)  =  n * factorial(n-1)     (if n>0)
```

3.9. Read the section "More on Notation (Optional)" and introduce new notation for the functions you defined above. The section is from Chapter 1 of

Logical Foundations (SF, vol. 1):
https://softwarefoundations.cis.upenn.edu/lf-current/Basics.html#lab41

3.10 Read the section "Fixpoints and Structural Recursion (Optional)" and solve the exercise listed in that section. The section is from Chapter 1 of *Logical Foundations* (SF, vol. 1):
https://softwarefoundations.cis.upenn.edu/lf-current/Basics.html#lab42

3.11. Solve the exercise on binary representations listed at the bottom of Chapter 1 of *Logical Foundations* (SF, vol. 1):
https://softwarefoundations.cis.upenn.edu/lf-current/Basics.html#lab57