Since this is the first sheet, and we find it hard to assess the difficulty of these problems, we will only count the best three of your (up to) four solutions, so that **30 points** is a perfect score.

1. **Construct a functionalal!** (10 points)
   Find an expression E fitting into the typing judgement

$$A : \mathtt{Type} \vdash E : ((A \to A) \to A) \to A$$

   and construct a proof tree for it. If you think of E as a program that computes something, what does it do?

   Explanation of the term 'functionalal': An expression of type $A \to A$ is a function; an expression of type $(A \to A) \to A$ is a functional; hence an expression of type $((A \to A) \to A) \to A$ is a functionalal! (Due to Jim Dolan and John Baez.)

2. **Associativity of function composition.** (10 points)
   In a context $A, B : \mathtt{Type}$, $f : A \to B$ and $g : B \to C$, recall that the composite function was defined as

$$g \circ f :\equiv \lambda x.g(f(x)) \; : \; A \to C.$$

   In the context

$$A, B, C, D : \mathtt{Type},\ f : A \to B,\ g : B \to C,\ h : C \to D,$$

   show that this composition is associative up to judgemental equality,

$$(h \circ g) \circ f \equiv h \circ (g \circ f) \; : \; A \to D,$$

   by constructing a suitable proof tree (not necessarily in all detail).

   Note: Since `Coq` takes care of judgemental equalities implicitly and automatically, this should be done by hand.

3. **Product of the unit type with itself.** (10 points)
   Construct functions $\mathbf{1} \to \mathbf{1} \times \mathbf{1}$ and $\mathbf{1} \times \mathbf{1} \to \mathbf{1}$. What happens when you compose them to a function $\mathbf{1} \to \mathbf{1}$ and to a function $\mathbf{1} \times \mathbf{1} \to \mathbf{1} \times \mathbf{1}$? Try to simplify the expression as much as possible via judgemental equality. You may start using a more informal mode of reasoning: proof trees are not required.

4. **De Morgan's laws.** (10 points)
   Formulate and prove the statement

   "if (not P) or (not Q), then not (P and Q)"

   in type theory.

   The converse implication "if not (P and Q), then (not P) or (not Q)" does not hold constructively. However, the classically equivalent statement

   "if not (P and Q), then P implies (not Q)"

   is still derivable. Formulate and prove this in type theory as well.