```
(** * Lecture 01 *)

Inductive bool : Set :=
  | true : bool
  | false : bool.

Definition notb (b: bool) : bool :=
  match b with
  | true => false
  | false => true
  end.

Definition andb (b1 : bool) (b2: bool) : bool :=
  match b1 with
  | true => b2
  | false => false
  end.

Lemma andb_comm:
  forall b1 b2,
  andb b1 b2 = andb b2 b1.
Proof.
  intro x.
  intro y.
  destruct x.
  - destruct y.
    + reflexivity.
    + simpl. reflexivity.
  - destruct y.
    + simpl. reflexivity.
    + reflexivity.
Qed.

Lemma andb_comm':
  forall b1 b2,
  andb b1 b2 = andb b2 b1.
Proof.
  destruct b1; destruct b2; auto.
Qed.

Check andb.

(** andb : bool -> (bool -> bool) *)

Check (andb true).

(** currying *)

Inductive nat : Set :=
| O : nat
| S : nat -> nat.

Definition isZero (n: nat) : bool :=
  match n with
  | O => true
  | S _ => false
  end.

Lemma isZero_O:
  isZero O = true.
Proof.
  simpl.
  reflexivity.
Qed.

Fixpoint add (n1: nat) (n2: nat) : nat :=
  match n1 with
  | O => n2
  | S m1 => S (add m1 n2)
```



```
  end.

Fixpoint ellis_add (n1: nat) (n2: nat) : nat :=
  match n1 with
  | O => n2
  | S m1 => add m1 (S n2)
  end.

Lemma O_add:
  forall n,
  add O n = n.
Proof.
  simpl.
  reflexivity.
Qed.

Lemma add_O:
  forall n,
  add n O = n.
Proof.
  intro x.
  induction x.
  - apply O_add.
  - simpl.
    rewrite IHx.
    reflexivity.
Qed.

Inductive list (A: Set) : Set :=
| nil : list A
| cons : A -> list A -> list A.

Fixpoint length (A: Set) (l: list A) : nat :=
  match l with
  | nil => O
  | cons x xs => S (length A xs)
  end.
```