In the standard library.

A set of data values.

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

```
Definition negb (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.

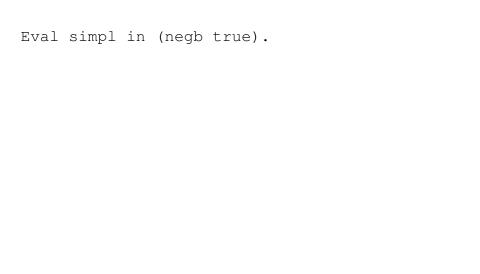
Example test_negation:
 (negb true) = false.

Proof. simpl. reflexivity. Qed.

- Use Eval on a test case and observe the result.

Haskell.

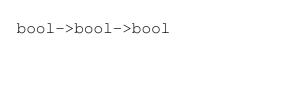
- Use Example/Theorem/whatever to record expected result, then as Cog to verify.
- "extract" function Definition to OCaml, Scheme, or



- andb
- -orb
- negb

admit fills in holes in Definitions.

Admitted fills in holes in proofs.



It causes Coq to print the type of an expression.

If you put declarations between Module X and End X then after End the definitions are referred to as X, foo.

Inductive nat : Type :=
 | 0 : nat
 | S : nat -> nat.

A set of *expressions*, inductively defined. The definition tells us exactly how members of the type can be constructed, and excludes all other expressions.

Functions come with *computation rules*. Data constructors have no behavior attached.

- Definition
- Fixpoint in case of recursion

Structural (or *primitive*) recursion. That means recursive calls must be on strictly smaller values, guaranteeing termination.

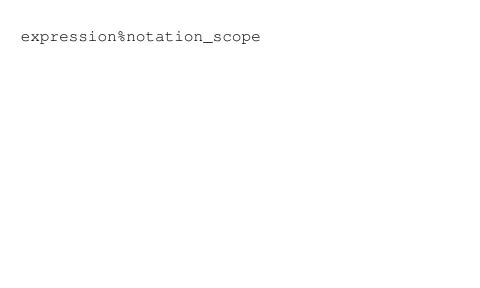
The following are equivalent:

```
(n m : nat)
(n: nat) (m: nat)
```

A comma is placed between then in the scrutinee and between the two sides of each matching pattern.

With ${\tt Notation}$ constructions which also define associativity and precedence.

- Numerals
- Operators
- Collections syntax





Simplifies both sides before testing (including by using simpl).

Among other things, reflexivity may expand definitions. simpl never will.

For a conditional it introduces the antecedent as an assumption. For a universally quantified statement it introduces an arbitrary element of the domain and discharges

the quantifier.