Where does Coq define booleans and numbers?

In the standard library.

What's a "type" in Coq?

A set of data values.

As an example of an enumerated type, define the boolean type.

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

Define a boolean negation function.

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

Define a function for boolean conjunction.

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

Make a named assertion that `~true` is `false`, then prove it.

```
Example test_negation:
   (negb true) = false.
Proof. simpl. reflexivity. Qed.
```

Name three ways to check that a function works.

- Use `Eval` on a test case and observe the result.
- Use `Example`/`Theorem`/whatever to record expected result, then as Coq to verify.
- "extract" function `Definition` to OCaml, Scheme, or Haskell.

Apply negation to the boolean `true`.

```
Eval simpl in (negb true).
```

What are Coq's names for boolean and/or/not?

- andb
- orb
- negb

How do you fill in a hole in a `Definition`? In an `Example`?

`admit` fills in holes in `Definitions`.
`Admitted` fills in holes in proofs.

How does Coq write the type of a boolean conjunction function?

```
bool->bool->bool
```

What does the `Check` command do?

It causes Coq to print the type of an expression.

How will we use the module system?

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

As an example of a type with a sum constructor, define `nat`.

```
Inductive nat : Type :=
  | O : nat
  | S : nat -> nat.
```

When we use `Inductive` to define a type, we should see it as what?

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

What is the fundamental difference between a data constructor and and functions?

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

Name some keywords that can introduce a function.

- `Definition`
- `Fixpoint` in case of recursion

What kind of recursion does Coq allow?

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

What notational convenience does Coq provide for multiple parameters of the same type?

The following are equivalent:

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

How does one match on *multiple* expressions?

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

How is "language support" introduced for some definitions?

Name two kinds of language support available.

With `Notation` constructions which also define associativity and precedence.

- Numerals
- Operators
- Collections syntax

How can one choose between multiple notation interpretations for an expression.

expression%notation_scope

Which tactic is like `simpl` "on steroids"?

compute

The `reflexivity` tactic implicitly does what?

What's the difference between the simplification of `simpl` and that of `reflexivity`?

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

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

What does the `intros` tactic do?

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.