The following two are equivalent:

```
pair x y
(x, y)
```

```
destruct p as (n, m). (* using introduced notation *)
destruct p as [n m].  (* sugarless syntax *)
```

List literals:

```
[]
[1, ..., n]
```

Cons:

```
el :: lst
```

`++` in place of `app`.

??

```
if X then Y else Z.

match X with
| true  => Y
| false => Z
end.
```

A multiset, an orderless collection in which elements may appear more than once.

Creating type aliases:

```
Definition bag := natlist.
```

The name of another `Definition`.

```
Definition sum : bag => bag -> bag := app.
```

... inductively defined datatype.

```
induction l as [| n l']
Case "l = nil".
  ...
Case "l = cons n l'".
  ...
```

```
SearchAbout foo
```

Prints all the theorems Coq knows that involve `foo`.

They are values as in other functinal languages, but they are more general. Instead of requiring booleans they require any datatype with two constructors with the first being truthy and the second falsy.

Overusing intros may lead to a weaker induction hypothesis when the `inductive` tactic is later used. The hypothesis will be on specific variables chosen by `intros`, instead of a hypothesis over universally quantified variables. You can always finish using `intros` after you have the more general hypothesis to work with.

- It allows you match the current goal to the conclusion of a conditional hypothesis of the current context. Like modus ponens in reverse.
- New subgoals will be generated for every premise in the conditional.
- Keep in mind a non-conditional statement can be viewed in this case as a degenerate conditional, allowing you to match the current subgoal to a hypothesis and end the proof.

Coq will attempt to bind the variables in the new premises-derived subgoals with the correct concrete terms. The conclusion of the theorem must match the goal *exactly*, but only after this binding is done.

It swaps the sides of the goal, assuming it's an equation.

`simpl`, although keep in mind `reflexivity`'s simplification is more complicated than just `simpl`.