What mathematical notation do we introduce for pairs?

The following two are equivalent:

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

Destruct a pair.

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

What notation do we introduce for lists?

List literals:

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

Cons:

```
el :: lst
```

What notation do we introduce for appending two lists?

`++` in place of `app`.

What does it mean to say that the `pair` constructor is surjective?

??

What's an alternative to `if/then/else` expressions.

```
if X then Y else Z.
```

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

What is a `bag`?

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

What's another use for `Definition`, apart from creating functions?

Creating type aliases:

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

Unlike in a Scala `def`, the RHS of a Coq `Definition` can simply be what?

The name of another `Definition`.

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

Induction can actually be done on any ...

... inductively defined datatype.

Demonstrate induction on lists.

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

What is the `SearchAbout` command?

```
SearchAbout foo
```

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

Describe conditionals in Coq.

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.

Why might one not use `intros` to the fullest extent possible?

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.

Describe the `apply` tactic.

- 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.

What happens when the `apply` statement is universally quantified?

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.

What is the `symmetry` tactic?

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

Like `reflexivity`, `apply` does what first?

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