The following two are equivalent:

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

```
destruct p as (n, m).
```

For whatever reason there are no brackets around the pattern.

List literals:

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

Cons:

```
el :: lst
```

++

default

```
if X then Y else Z.

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

Creating type aliases:

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

The name of another `Definition`.

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

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

```
SearchAbout foo
```

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

It allows you match the current goal to the conclusion of a conditional hypothesis of the current context. Like modus ponens in reverse. 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.