Today we will learn...

- What are proofs?
- Logical connectives
- Inductive propositions
What are proofs?
What is a type? What is a value?

- `nat` is a type
What is a type? What is a value?

- `nat` is a type
- `5` is a value of type `nat`
What is a type? What is a value?

- nat is a type
- 5 is a value of type nat
- Notations 5 : nat means 5 has type nat
What is a type? What is a value?

- nat is a type
- 5 is a value of type nat
- Notations $5 : \text{nat}$ means 5 has type nat
- *Types can be thought of as sets*
  - $5 : \text{nat}$ a programming notation $5 \in \mathbb{N}$
Exercise

Consider the following Coq excerpt:

```coq
Definition x := 5.
```

- What is $x$?
Exercise

Consider the following Coq excerpt:

```
Definition x := 5.
```

- What is $x$? A variable.
- What is the value of $x$?
Exercise

Consider the following Coq excerpt:

```
Definition x := 5.
```

- What is `x`? A variable.
- What is the value of `x`? 5
- What is the type of `x`?
Exercise

Consider the following Coq excerpt:

Definition x := 5.

- What is x? A variable.
- What is the value of x? 5
- What is the type of x? nat
- How do I query the type of x in Coq?
Exercise

Consider the following Coq excerpt:

\[
\text{Definition } x := 5. \\
\]

- What is \(x\)? A variable.
- What is the value of \(x\)? \(5\)
- What is the type of \(x\)? \(\text{nat}\)
- How do I query the type of \(x\) in Coq? Using \text{Check}.
- How do I query the value of \(x\) in Coq?
Exercise

Consider the following Coq excerpt:

```
Definition x := 5.
```

- What is \( x \)? A variable.
- What is the value of \( x \)? 5
- What is the type of \( x \)? nat
- How do I query the type of \( x \) in Coq? Using Check.
- How do I query the value of \( x \) in Coq? Using Print.
What is a proof? What is a proposition?

- A proof (or a proof object): a completed proof of some goal
  - usually written using tactics
  - a proof object is a value of a proposition
What is a proof? What is a proposition?

- **A proof** (or a proof object): a \textit{completed} proof of some goal
  - usually written using tactics
  - a proof object is a \textit{value} of a proposition
- **A proposition**: is a formula written in some logic
  - Propositions are of type \textit{Prop}
  - You can confirm that something is a proposition using \texttt{Check}
What is a proof? What is a proposition?

- **A proof** (or a proof object): a completed proof of some goal
  - usually written using tactics
  - a proof object is a **value** of a proposition

- **A proposition**: is a formula written in some logic
  - Propositions are of type `Prop`
  - You can confirm that something is a proposition using `Check`

- **A truthful proposition**: a proposition that contains a proof
  - `Proof : Proposition`
  - We also say that the proposition **holds** (if there is some proof of it)
What is a proof? What is a proposition?

- **A proof** (or a proof object): a *completed* proof of some goal
  - usually written using tactics
  - a proof object is a *value* of a proposition
- **A proposition**: is a formula written in some logic
  - Propositions are of type \texttt{Prop}
  - You can confirm that something is a proposition using \texttt{Check}
- **A truthful proposition**: a proposition that contains a proof
  - \texttt{Proof: Proposition}
  - We also say that the proposition **holds** (if there is some proof of it)
- **Assumption**: a synonym of a proof
What is a proof? What is a proposition?

- **A proof** (or a proof object): a *completed* proof of some goal
  - usually written using tactics
  - a proof object is a **value** of a proposition

- **A proposition**: is a formula written in some logic
  - Propositions are of type Prop
  - You can confirm that something is a proposition using Check

- **A truthful proposition**: a proposition that contains a proof
  - Proof : Proposition
  - We also say that the proposition **holds** (if there is some proof of it)

- **Assumption**: a synonym of a proof

- **Proof state**: zero or more assumptions and 1 or more goals we need to prove
  - Each assumption is an implication to the current goal
  - Each sub-goal is a conjunctions
Exercise

- Is $10$ a proposition?
Exercise

- Is $10$ a proposition? No. $10$ is a natural number.
- Is $2 = 2$ a proposition?
Exercise

- Is 10 a proposition? No. 10 is a natural number.
- Is $2 = 2$ a proposition? Yes.
- Is Nat.eqb 2 2 a proposition?
Exercise

- Is \(10\) a proposition? No. \(10\) is a natural number.
- Is \(2 = 2\) a proposition? Yes.
- Is \(\text{Nat.eqb } 2\) \(2\) a proposition? No, \(\text{Nat.eqb } 2\) \(2\) is an expression of type \(\text{bool}\).
- Is the code below a proposition?

```latex
Lemma example: 2 = 2.
Proof.
   reflexivity.
Qed.
```

No, the code above is a **proof** of formula \(2 = 2\).

- What is `example`?
Exercise

- Is 10 a proposition? No. 10 is a natural number.
- Is 2 = 2 a proposition? Yes.
- Is Nat.eqb 2 2 a proposition? No, Nat.eqb 2 2 is an expression of type bool.
- Is the code below a proposition?

```
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a proof of formula 2 = 2.

- What is example? A proof of 2 = 2.
- What is the value of example?
Exercise

- Is $10$ a proposition? No. $10$ is a natural number.
- Is $2 = 2$ a proposition? Yes.
- Is `Nat.eqb 2 2` a proposition? No, `Nat.eqb 2 2` is an expression of type `bool`.
- Is the code below a proposition?

```coq
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a **proof** of formula $2 = 2$.

- What is `example`? A proof of $2 = 2$.
- What is the value of `example`? `reflexivity`. (actually `eq_refl`)
- What is the type of `example`?
Exercise

- Is $10$ a proposition? No. $10$ is a natural number.
- Is $2 = 2$ a proposition? Yes.
- Is $\text{Nat.eqb} \ 2 \ 2$ a proposition? No, $\text{Nat.eqb} \ 2 \ 2$ is an expression of type $\text{bool}$.
- Is the code below a proposition?

```
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a proof of formula $2 = 2$.

- What is example? A proof of $2 = 2$.
- What is the value of example? reflexivity. (actually eq_refl)
- What is the type of example? $2 = 2$.
- What is the type of $2 = 2$?
Exercise

- Is **10** a proposition? No. **10** is a natural number.
- Is **2 = 2** a proposition? Yes.
- Is **Nat.eqb 2 2** a proposition? No, **Nat.eqb 2 2** is an expression of type **bool**.
- Is the code below a proposition?

```lean
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a **proof** of formula **2 = 2**.

- **What is example?** A proof of **2 = 2**.
- **What is the value of example?** **reflexivity.** (actually eq_refl)
- **What is the type of example?** **2 = 2**.
- **What is the type of 2 = 2?** **Prop.**
Exercise

- Is 10 a proposition? No. 10 is a natural number.
- Is 2 = 2 a proposition? Yes.
- Is Nat.eqb 2 2 a proposition? No, Nat.eqb 2 2 is an expression of type bool.
- Is the code below a proposition?

```lean
Lemma example: 2 = 2.
Proof.
  reflexivity.
Qed.
```

No, the code above is a proof of formula 2 = 2.

- What is example? A proof of 2 = 2.
- What is the value of example? reflexivity. (actually eq_refl)
- What is the type of example? 2 = 2.
- What is the type of 2 = 2? Prop.
Inductive propositions

We have seen how to define types inductively; propositions can also be defined inductively.

- instead of Type we use Prop
- the parameters are not just values, but propositions
- the idea is to build your logical argument as structured data

We will now encode various logical connectives using inductive definitions.
Conjunction

\[ P \land Q \]
What is $P \land Q$?

1. What is the type of $P$?
What is $P \land Q$?

1. What is the type of $P$? Prop
2. What is the type of $Q$?
What is $P \land Q$?

1. What is the type of $P$? Prop
2. What is the type of $Q$? Prop
3. What is the type of $\land$?
What is $P \land Q$?

1. What is the type of $P$? Prop
2. What is the type of $Q$? Prop
3. What is the type of $\land$? Prop $\rightarrow$ Prop $\rightarrow$ Prop
What is $P \land Q$?

Let and represent $\land$:

$\text{and}: \text{Prop} \to \text{Prop} \to \text{Prop}$

Recall how we defined a pair:

$\text{Inductive pair (X:Type) (Y:Type) : Type := ...}$

How would we define and?
Conjunction

Inductive and (P Q : Prop) : Prop :=
| conj : P → Q → and P Q.

• apply `conj` to solve a goal, inversion in a hypothesis
• The `/\` operator represents a logical conjunction (usually typeset with `∧`)
• The split tactics is used to prove a goal of type `?X /\ ?Y`, where `?X` and `?Y` are propositions

Notice that `P /\ Q` is a type (a proposition) and that `conj` is the only constructor of that type.
Conjunction example

Example and_example : 3 + 4 = 7 /\ 2 * 2 = 4.
Proof.

apply conj.

(Done in class.)
Conjunction example 1

More generally, we can show that if we have propositions $A$ and $B$, we can conclude that we have $A \land B$.

Goal for all $A$, $B$ : Prop, $A \rightarrow B \rightarrow A \land B$. 
Conjunction in the hypothesis

Example and_in_conj:

\[
\text{forall } x \ y, \\
3 + x = y \land 2 \cdot 2 = x \rightarrow \\
x = 4 \land y = 7.
\]

Proof.

\[
\text{intros } x \ y \ H\text{conj.} \\
\text{destruct } H\text{conj as [Hleft Hright].}
\]

(Done in class.)
Lemma correct_2 : \forall A \ B : \text{Prop}, A \land B \rightarrow A.
Proof.

Lemma correct_3 : \forall A \ B : \text{Prop}, A \land B \rightarrow B.
Proof.

(Done in class.)
Disjunction

\[ P \lor Q \]
What is $P \lor Q$?

1. What is the type of $P$?
What is $P \lor Q$?

1. What is the type of $P$? Prop
2. What is the type of $Q$?
What is $P \lor Q$?

1. What is the type of $P$? Prop
2. What is the type of $Q$? Prop
3. What is the type of $\lor$?
What is $P \lor Q$?

1. What is the type of $P$? Prop
2. What is the type of $Q$? Prop
3. What is the type of $\lor$? Prop $\rightarrow$ Prop $\rightarrow$ Prop

How can we define an disjunction using an inductive proposition?
Disjunction

\[ \text{Inductive or } (A \cdot B : \text{Prop}) : \text{Prop} := \]
\[ | \text{or_introl } : A \rightarrow \text{or} \cdot A \cdot B \]
\[ | \text{or_intror } : B \rightarrow \text{or} \cdot A \cdot B \]

- apply \text{or_introl} or apply \text{or_intror} to goal; inversion to hypothesis
- The \( \lor \) operator represents a logical disjunction (usually typeset with \( \lor \))
- The left (right) tactics are used to prove a goal of type \( ?X \lor ?Y \), replacing it with a new goal \( ?X \) (\( ?Y \) respectively)
Disjunction example

**Theorem or_1:** \( \forall A B : \text{Prop}, \ A \rightarrow A \lor B. \)

**Theorem or_2:** \( \forall A B : \text{Prop}, \ B \rightarrow A \lor B. \)

*(Done in class.)*
Disjunction in the hypothesis

Tactics **destruct** can break a disjunction into its two cases. Tactics **inversion** also breaks a disjunction, but leaves the original hypothesis in place.

**Lemma** or_example :
\[
\forall n \ m : \text{nat}, \quad n = 0 \lor m = 0 \implies n \times m = 0.
\]

**Proof.**
- intros \( n \ m \) Hor.
- destruct Hor as [Heq | Heq].
Recall a proof state

1 subgoal
T : Type
x : T
P : Prop
H1 : 1 = x
H2 : P

All hypothesis are **variables** of a specific type, Type, or proposition
Goals are (usually) propositions
**Propositions** (instances of Prop) can mention **values**

Can a proposition mention **pair**, the constructor of prod? Can a proposition mention **conj**, the constructor of and?
Recall a proof state

1 subgoal
T : Type
x : T
P : Prop
H1 : 1 = x
H2 : P

1 = 2 /\ P

- All hypothesis are **variables** of a specific type, Type, or proposition
- Goals are (usually) propositions
- **Propositions** (instances of Prop) can mention **values**

Can a proposition mention **pair**, the constructor of prod? Can a proposition mention **conj**, the constructor of and? Yes and no, respectively.
Theorem and_conj: \( \forall P \; Q : \text{Prop}, \)
\[ P \rightarrow Q \rightarrow P \land Q. \]

Proof.
  \begin{itemize}
  \item intros P Q H1 H2.
  \item apply conj.
  \item apply H1.
  \item apply H2.
  \end{itemize}
Qed.
Theorems are expressions too

**Theorem** and\_conj: \( \forall P \ Q: \text{Prop}, \quad P \rightarrow Q \rightarrow P \land Q. \)

**Proof.**
- intros \( P \ Q \ H1 \ H2. \)
- apply (conj \( H1 \ H2 \)).

Qed.

Proposition-constructors and theorems are **functions** whose parameters are **evidences**.
Truth
Truth

Truth can be encoded in Coq as a proposition that always holds, which can be described as a proposition type with a single constructor with 0-arity.

\[ \text{Inductive True : Prop := I : True.} \]

You will note that proposition \text{True} is not a very useful one.
Truth example

Goal True.

(Done in class.)
Falsehood
So far we only seen results that are provable (eg, plus is commutative, equals is transitive)

How to encode falsehood in Coq?
Falsehood in Coq is represented by an **empty** type.

```coq
Inductive False : Prop :=.
```

- The only way to reach it is by using the exploding principle
- **No constructors available.** Thus, no way to build an inhabitant of False.
Example:

Goal $1 = 2 \rightarrow$ False.

Goal False $\rightarrow$ $1 = 2$.

Goal False.

*(Done in class.)*
Negation

\neg P
The negation of a proposition $\neg P$ is defined as

\begin{verbatim}
(* As defined in Coq's stdlib *)
Definition not (H:Prop) := H -> False.
\end{verbatim}

Goal not (1 = 2).

Outputs:
1 subgoal
---------------------------------(1/1)
1 <= 2
(Done in class.)
Negation-related notations

- \text{not } P \text{ is the same as } \sim P, \text{ typeset as } \neg P
- \text{not } (x = y) \text{ is the same as } x \not= y, \text{ typeset as } x \neq y

Can we rewrite not with an inductive proposition?