

# LEAN 4 CHEATSHEET

If a tactic is not recognized, write `import Mathlib.Tactic` at the top of your file.

Logical symbol	Appears in goal	Appears in hypothesis
$\forall$ (for all)	<code>intro x</code>	<code>apply h</code> or <code>specialize h x</code>
$\rightarrow$ (implies)	<code>intro h</code>	<code>apply h</code> or <code>specialize h1 h2</code>
$\neg$ (not)	<code>intro h</code>	<code>apply/specialize/contradiction</code>
$\leftrightarrow$ (if and only if)	<code>constructor</code>	<code>rw [h]/rw [← h]/apply h.mp/apply h.mpr</code>
$\wedge$ (and)	<code>constructor</code>	<code>rcases h with ⟨h1, h2⟩</code>
$\exists$ (there exists)	<code>use x</code>	<code>rcases h with ⟨x, hx⟩</code>
$\vee$ (or)	<code>left or right</code>	<code>rcases h with h h</code>

Tactic	Effect
<code>exact <i>expr</i></code>	prove the current goal exactly by <i>expr</i> .
<code>apply <i>expr</i></code>	prove the current goal by applying <i>expr</i> to some arguments.
<code>refine <i>expr</i></code>	like <code>exact</code> , but <i>expr</i> can contain sub-expressions <code>?_</code> that will be turned into new goals.
<code>convert <i>expr</i></code>	prove the goal by showing that it is equal to the type of <i>expr</i> .
<code>have h : <i>proposition</i> := <i>expr</i></code>	add a new hypothesis <i>h</i> of type <i>proposition</i> .
<code>have h : <i>proposition</i></code>	... also creates <i>proposition</i> as a new goal.
<code>by_cases h : <i>proposition</i></code>	create two goals, one where <i>h</i> is the hypothesis that <i>proposition</i> is true and one where <i>h</i> is the hypothesis where it is false.
<code>congr</code>	prove an equality using congruence rules
<code>gcongr</code>	prove an inequality using congruence rules
<code>exfalso</code>	replace the current goal by <code>False</code> .
<code>by_contra h</code>	start a proof by contradiction, where <i>h</i> is the hypothesis that the current goal is false.
<code>push_neg</code>	push negations into other connectives in the goal (or in <i>h</i> ); e.g. change $\neg \forall x, P x$ to $\exists x, \neg P x$ .
<code>push_neg at h</code>	
<code>rw [<i>expr</i>]</code>	in the goal, replace (all occurrences of) the left-hand side of <i>expr</i> by its right-hand side. <i>expr</i> must be an equality or if and only if statement.
<code>rw [←<i>expr</i>]</code>	... rewrites from right-to-left
<code>rw [<i>expr</i>] at h</code>	... rewrite in hypothesis <i>h</i>
<code>simp</code>	simplify the goal using all lemmas tagged <code>@[simp]</code> .
<code>simp at h</code>	... simplify in hypothesis <i>h</i> .
<code>simp [* , <i>expr</i>]</code>	... also simplify with all hypotheses and <i>expr</i> .
<code>simp only [<i>expr</i>]</code>	... do not simplify with all standard lemmas, only with <i>expr</i> .
<code>simp?</code>	... ask Lean to generate a <code>simp only [...]</code> tactic that applies the same simplifications.
<code>exact?</code>	search for a single lemma that closes the goal using the current hypotheses.
<code>apply?</code>	gives a list of lemmas that can apply to the current goal.
<code>rw?</code>	gives a list of lemmas that can be used to rewrite the current goal.
<code>linarith</code>	prove linear (in)equalities from the hypotheses
<code>ring / noncomm_ring</code>	prove the goal by using the axioms of a commutative ring / ring / abelian group / group.
<code>abel / group</code>	
<code>aesop</code>	simplify the goal, and use various techniques to prove the goal.
<code>tauto</code>	prove certain goals using first-order logic.