An Introduction to Formal Real Analysis, Rutgers University, Fall 2025, Math 311H

Lecture 1: The Story of Real Analysis

Prof. Alex Kontorovich

This text is automatically generated by LLM from "Real Analysis, The Game", Lecture 1

1 Preamble

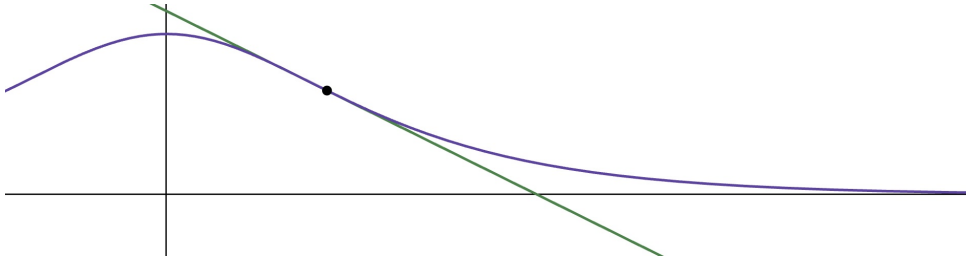
SIMPLICIO: What is "Real Analysis"?

SOCRATES: Oh, it's just Calculus, but done "right".

SIMPLICIO: Huh? Why does Calculus need a new name? What's wrong with it?

SOCRATES: Well, nothing really. Quick: what's a derivative?

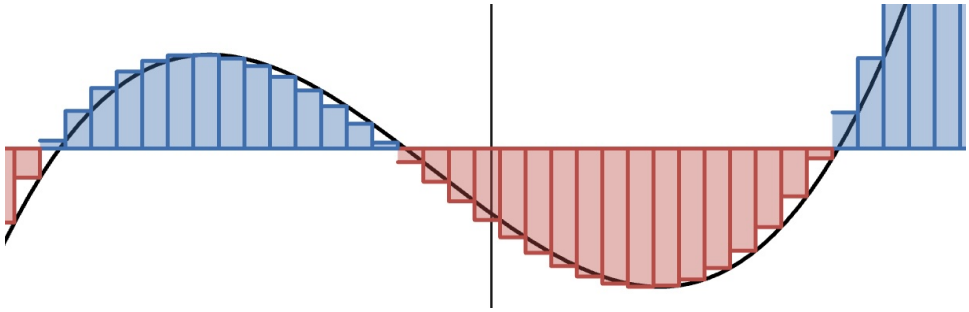
SIMPLICIO: Easy! If I have a function $f : \mathbb{R} \to \mathbb{R}$ and it's differentiable at x, then the derivative is $f'(x) := \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$. This represents the "instantaneous" slope of the graph of the function $y = f(x)$ at the point $(x, f(x))$.



SOCRATES: Very good! And tell me please, what is an integral?

SIMPLICIO: That's easy, too! If you want to integrate our function f along an interval, $[a, b]$, say, you pretend that you have infinitely many, infinitely small rectangles, work out their areas as base times height, and add them up:

$$\int_a^b f(x)dx := \lim_{N\to\infty} \sum_{j=1}^{N} \frac{b-a}{N} f\left(a + j\frac{b-a}{N}\right)$$



SOCRATES: Great. And what are the two Fundamental Theorems of Calculus?

SIMPLICIO: These too are easy! The first one says that if you make a new function by integrating f up to a variable amount, x, that is, let

$$F(x) := \int_a^x f(t)dt$$

then the derivative of the new function is just $F'(x) = f(x)$.

SOCRATES: And the second?

SIMPLICIO: The second one says that, conversely, if F is an antiderivative of f, that is, $F'(x) = f(x)$, then it's easy to work out the area under the curve, because

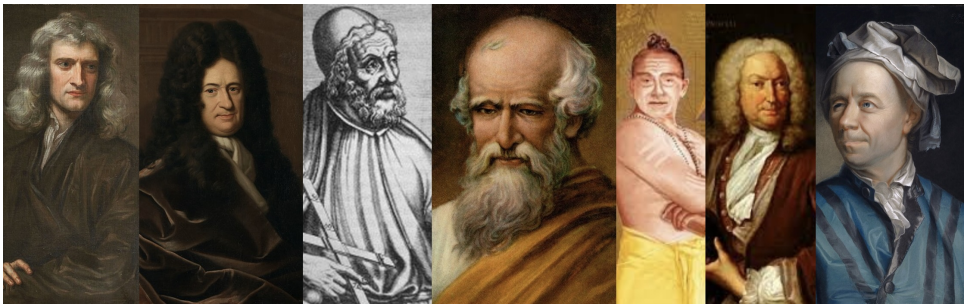
$$\int_a^b f(x)dx = F(b) - F(a)$$

So differentiation and integration are inverse operations!

SOCRATES: Perfect. Now, here's the problem. You used words like "limit", "infinitely many", "infinitely small", and so on. What do they *actually* mean?

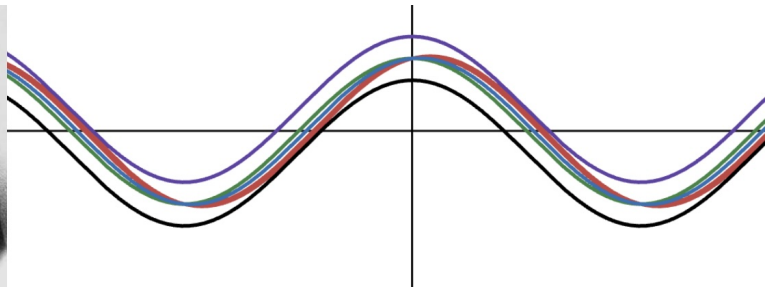
SIMPLICIO: Oh, you know, it's when something happens "eventually". You just have to get used to the feel of it.

SOCRATES: Hmm yes, I see. I agree that that's an OK way to think of it, for a while at least, and one that suited Newton (who was quite uncomfortable with such words), and Leibniz (who was more care-free here), the two 17th century inventors of calculus (if you don't count people like the ancient Greeks Eudoxus and Archimedes, or the 14th century Indian Madhava... but this isn't a history lesson). Leibniz taught the Bernoulli brothers (the world's "first AP Calc students"!), who taught, among others, the Marquis de l'Hopital, and the great Leonhard Euler (the first "Calc native"), who taught the rest of us. All was going quite well... and then came the 19th Century.



SIMPLICIO: Huh? What happened in the 19th Century?

SOCRATES: Well you see, a guy named Augustin-Louis Cauchy came along (roughly in the 1810s), and started making a fuss that we weren't really doing things perfectly "rigorously". He set out to reprove the theorems of calculus using precise definitions rather than hand-waving. He was making great progress, including proving statements like: the limit of continuous functions is continuous.



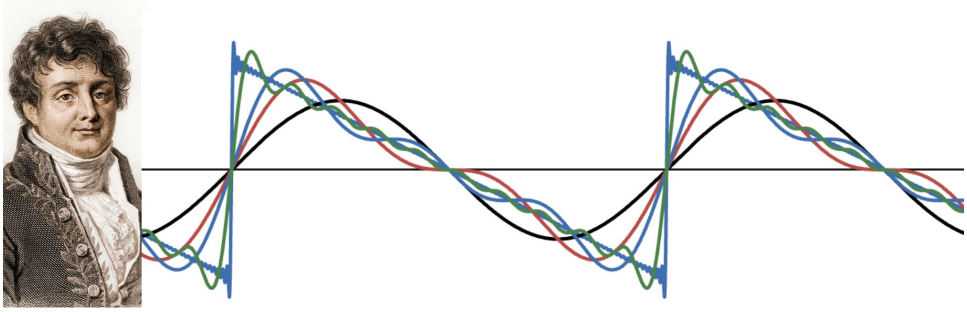
SIMPLICIO: Sure, that sounds perfectly reasonable. A limit is a continuous process, so you do that to continuous functions, and of course in the end you should get something continuous, too. No?

SOCRATES: Well, the problem is that around the same time, a guy named

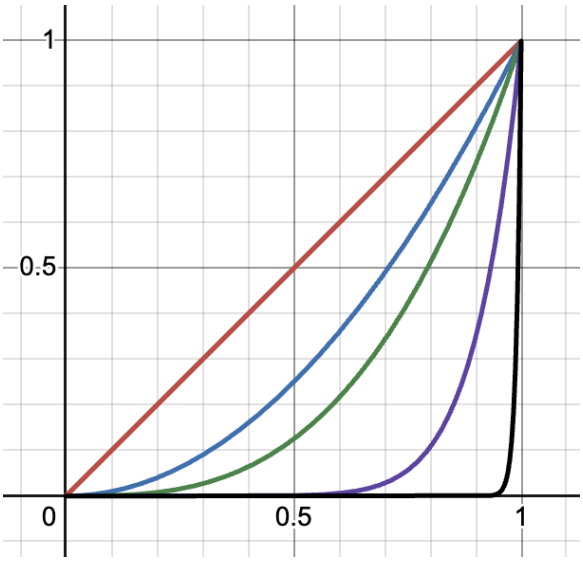
Joseph Fourier was going around claiming that he could add up a bunch of sines and cosines, and get basically any function he wants, including, say, the discontinuous sawtooth!

SIMPLICIO: What?!

SOCRATES: Look for yourself: Here's a graph of $\sum_{n=1}^{100} \frac{1}{n} \sin(nx)$. As you take 100 out to infinity, Fourier claims that this will get closer and closer to a sawtooth function!



SIMPLICIO: Whoa. Wait, I can think of an even easier example: just look at the simplest family of polynomials, $f_n(x) = x^n$, on the unit interval $[0, 1]$. When you take high powers of any point strictly less than 1, that goes to 0 in the limit, but powers of 1 itself always stay at 1. So the limiting function is discontinuous, too! What the heck is going on here?

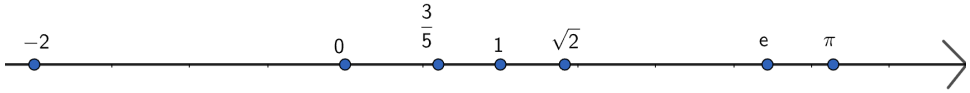


SOCRATES: Very good, Simplicio! Exactly right, between Fourier and Cauchy, they "broke math". You break it, you buy it!

SIMPLICIO: Ok, so what's the right answer, how *do* you do calculus rigorously?

SOCRATES: Not so fast! Things got even worse, and by the mid-19th century, people realized that we don't even know what the real numbers *are*!

SIMPLICIO: What? What do you mean, what are they? Here they are right here: There's zero, and one, and -2, and $\frac{3}{5}$, and $\sqrt{2}$, and e and π. What's the problem?



SOCRATES: Well, do you remember that you need something called the Intermediate Value Theorem in calculus?

SIMPLICIO: Sure, if you have a continuous function, and it goes from being negative to being positive, then it has to cross zero at some point in between.

SOCRATES: Very good. Tell me about the function $f : x \mapsto x^2 - 2$. (We say: "f maps x to $x^2 - 2$". Note that we write $f : \mathbb{R} \to \mathbb{R}$ to mean that f takes real numbers to real numbers, but replace $\to$ with $\mapsto$ (see the tail on the second arrow?) when we want to say what happens to a particular input x under the action of f. Let me remind you that x here is a dummy variable, so it's the same function if we'd said $f : u \mapsto u^2 - 2$.) In particular, what happens to f on the rational numbers?

SIMPLICIO: Ok, well if x is a rational number, then so is x^2, and hence so is $x^2 - 2$. So actually, we could say that $f : \mathbb{Q} \to \mathbb{Q}$, that is, f maps rational numbers to rational numbers. Over the reals, the graph of $y = f(x)$ is a simple parabola. But you'd asked me about the Intermediate Value Theorem. Hmm. When $x = 0$, I know that $f(x)$ will be $f(0) = 0^2 - 2 = -2$ which is negative. And when $x = 2$, $f(2) = 2^2 - 2 = 2$ which is positive.

SOCRATES: Go on...

SIMPLICIO: So there's a root of f somewhere between 0 and 2. But the place where f crosses the x-axis is at $x = \sqrt{2} \approx 1.41\ldots$.

SOCRATES: And what did the Pythagoreans know about this number?

SIMPLICIO: Supposedly one of them, Hippasus, figured out that $\sqrt{2}$ is irrational, which ruined their entire theory of geometry and form (they originally believed that *all* numbers were rational); legend has it that Hippasus

was drowned at sea for his heresy.

SOCRATES: So...

SIMPLICIO: So wait, if we just restrict to rational inputs, then this parabola is negative, and then it's positive, and it *never* crosses zero?! But there's tons of rational numbers almost everywhere you look. So what makes the real numbers different from the rational numbers, so that the Intermediate Value Theorem actually holds?

SOCRATES: Ah! Now, my friend, we are ready to begin.

2 Introduction to Lean

2.1 Theorem Prover Software

In this course, we will be using a "proof assistant" called Lean. This is software that checks that our proofs prove *exactly* what we claim they prove. It has other really cool pedagogical features that we'll get to later. It will take a little while to get used to the syntax, so until we're comfortable, we'll intersperse exercises teaching Lean with exercises teaching Real Analysis. Pretty soon all the exercises will just be about Real Analysis.

For this first exercise, we have a hypothesis that we called `h` (but we could've called it anything, like `x_eq_5`, or `Alice`) that says a real number `x` equals 5. Our goal is to prove that `x` equals 5. This shouldn't be very hard, but if you don't know the command, you'll be out of luck. Our goal is to prove *exactly* the same statement as one of the hypotheses. To solve that goal, the syntax is to write `exact`, then a space, and then the name of the hypothesis which exactly matches the goal.

Remark. The `exact` tactic solves a goal when one of the hypotheses is *exactly* the same as the goal. The syntax is `exact hypothesis_name`.

Theorem 2.1. *If we know that $x = 5$, then we can prove that $x = 5$.*

```
Statement (x : ℝ) (h : x = 5) : x = 5 := by
  exact h
```

Hint: Write `exact h` since the hypothesis `h` is exactly what we want to prove.

2.2 Conclusion

Perfect! You've completed your first Lean proof involving real numbers.

Remember: the `exact` tactic is used when you have exactly what you need to prove the goal. Look at the top right: your list of tactics now includes `exact`, and if you forget how it works or what it does, just click on it for a reminder.

3 The rfl tactic

3.1 When things are identical to themselves

Sometimes in mathematics, we need to prove that something equals itself. For example, we might need to prove that $x^2 + 2y = x^2 + 2y$.

This isn't quite the same as our previous exercise. There, we had a hypothesis `h` that told us `x = 5`, and we used `exact h` to prove the goal `x = 5`.

But now we don't have any hypothesis that says `x^2 + 2*y = x^2 + 2*y`. We're just being asked to prove that some expression equals itself. We can't say `exact something` because there's no `something`.

Instead, we will use what mathematicians call the *reflexive property* of equality: everything is equal to itself. In Lean, if you get to a situation where you're trying to prove an equality, and the two things on both sides are *identical*, then the syntax is to give the command `rfl` (short for "reflexivity").

Try it out!

Remark. The `rfl` tactic proves goals of the form `A = A` where both sides are *identical*.

Theorem 3.1. *Every mathematical expression equals itself.*

```
Statement (x y : ℝ) : x^2 + 2*y = x^2 + 2*y := by
  rfl
```

Hint: Write `rfl` since we're proving that something equals itself.

3.2 Conclusion

Excellent! You've learned the `rfl` tactic.

The key difference:

- Use `exact hypothesis_name` when you have a hypothesis that matches your goal exactly
- Use `rfl` when you need to prove that something equals itself

These are two of the most fundamental tactics in Lean. As we progress through real analysis, you'll see that many complex proofs ultimately come down to showing that two expressions are identical.

4 The rw tactic

4.1 Rewriting with equalities

Now let's learn about rewriting. Suppose you have a hypothesis called `Bob : x = 2`, and your goal is to prove that `x + y = 2 + y`.

Can you use `rfl`? No, because the two sides of the goal (`x + y` and `2 + y`) are not *identically* the same.

Can you use `exact Bob`? No, because `Bob` says `x = 2`, which is not *exactly* what the goal is asking for.

But you can use the hypothesis `Bob` to *rewrite* the goal. Since `Bob` tells us that `x = 2`, we can replace `x` with `2` in our goal.

In Lean, if you have a hypothesis which is an equality, and you want to replace the *left hand side* of that equality with the *right hand side* in your goal, you use the `rewrite` tactic. The syntax is:

`rewrite [hypothesis_name]`

Unfortunately, those square brackets are part of the Lean syntax, and there's nothing you or I can do about them right now. Just remember: `rewrite [Bob]` means "use the equality in `Bob` to rewrite the goal."

After you rewrite, you're not done. But you should know how to finish from there.

Try it out!

Remark. The `rewrite` tactic replaces the left-hand side of an equality with the right-hand side in the goal. The syntax is `rewrite [hypothesis_name1, hypothesis_name2, etc]`.

Theorem 4.1. *If we know that $x = 2$, then we can prove that $x + y = 2 + y$.*

```
Statement (x y : ℝ) (Bob : x = 2) : x + y = 2 + y := by
  rewrite [Bob]
  rfl
```

Hint: Type `rewrite [Bob]` to replace `x` with `2` in the goal. Then what?

4.2 Conclusion

Great! You've learned the `rewrite` tactic.

Notice what happened: after you typed `rewrite [Bob]`, the goal changed from `x + y = 2 + y` to `2 + y = 2 + y`. Then you needed to type `rfl` to finish the proof, since both sides were now identical.

So far you've learned:

- `exact hypothesis_name` when a hypothesis exactly matches your goal
- `rfl` when you need to prove something equals itself
- `rewrite [hypothesis_name]` when you want to use an equality to rewrite your goal

The `rewrite` tactic is incredibly powerful and you'll use it constantly in real analysis!

5 The ring_nf tactic

5.1 Algebraic manipulations

Now let's learn about algebraic simplification. Suppose you need to prove that $(x + y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$.

This is true by the basic laws of algebra - expanding the left side using the distributive law, commutativity, associativity, etc. But doing this by hand would be extremely tedious.

Fortunately, Lean has a powerful tactic called `ring_nf` ("ring normal form") that can automatically perform algebraic manipulations like:

- Expanding products
- Collecting like terms
- Rearranging using commutativity and associativity
- Applying the distributive law

When you have an algebraic identity involving addition, subtraction, and multiplication, `ring_nf` can often prove it automatically.

Try it out on this classic binomial expansion!

Remark. The `ring_nf` tactic puts both sides of an equation into a standard algebraic form and checks if they're equal.

Theorem 5.1. *The binomial expansion:* $(x + y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$.

```
Statement (x y : ℝ) : (x + y)^3 = x^3 + 3*x^2*y + 3*x*y
   ^2 + y^3 := by
  ring_nf
```

Hint: Write `ring_nf` to expand and simplify both sides algebraically.

5.2 Conclusion

Excellent! You've learned the `ring_nf` tactic.

This tactic is incredibly powerful for algebraic manipulations. It automatically handles all the tedious algebra that would take many steps to do by hand.

Your toolkit now includes:

- `exact hypothesis_name` for when a hypothesis exactly matches your goal
- `rfl` for proving something equals itself
- `rewrite [hypothesis_name]` for rewriting using equalities
- `ring_nf` for algebraic simplifications and expansions

As we move into real analysis proper, you'll find that `ring_nf` is invaluable for dealing with polynomial expressions, which appear everywhere in calculus!

6 The use tactic

6.1 Proving existence

Sometimes in mathematics, you need to prove that something exists. For example, suppose I wanted to ask you what the binomial coefficient in front of x^2y^2 is in the expansion of $(x+y)^4$; how would I do it? Lean can't ask questions, it can only prove theorems! So the way I would ask this is: prove that there exists a real number c such that

$$(x+y)^4 = x^4 + 4x^3y + cx^2y^2 + 4xy^3 + y^4.$$

The way to prove that such a number exists is to exhibit it, that is, tell me which number to *use*, and then prove that that number indeed satisfies the equation.

This is called an *existential statement*. In Lean, as in mathematics, existence is written using ∃ (read: "there exists"). This symbol is called the *existential quantifier*, and is written in Lean by typing `\exists`, that is, a backslash, then the word `exists`, and then a space. So this goal would look in Lean like so:

```
∃ (c : ℝ), (x + y)^4 = x^4 + 4*x^3*y + c*x^2*y^2 + 4*x*y^3 + y^4
```

To prove an existence statement, you need to provide a specific value that works. This is where the `use` tactic comes in.

If you think you know what value of `c` would work, you can write `use 42` (or with `42` replaced by whatever number you think is right). Lean will then substitute that value and ask you to prove that the resulting equation is true.

Try writing `use`, then a space, and then a number. Do you see what to do after that?

Remark. The `use` tactic provides a specific value to prove an existence statement.

Theorem 6.1. *There exists a real number that makes this binomial expansion work.*

```
Statement (x y : ℝ) : ∃ (c : ℝ), (x + y)^4 = x^4 + 4*x
    ^3*y + c*x^2*y^2 + 4*x*y^3 + y^4 := by
  use 6
  ring_nf
```

Hint: Write `use 42`, but with `42` replaced by the correct answer. Then how should you finish?

6.2 Conclusion

Perfect! You've learned the `use` tactic for existence proofs.

Notice what happened:

1. `use 6` told Lean that $c = 6$ is our proposed value
2. The goal changed to proving $(x + y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$
3. `ring_nf` verified that this algebraic identity is correct

The `use` tactic is fundamental in real analysis. You'll need it to:

- Find specific values of ε and δ in limit proofs
- Construct witnesses for existence theorems
- Provide counterexamples

Your growing toolkit:

- `exact`, `rfl`, `rewrite` for basic equality reasoning
- `ring_nf` for algebraic manipulation
- `use` for existence proofs

7 The intro tactic

7.1 Universal statements

In mathematics, we often need to prove statements that are true "for all" values of some variable. For example, we might want to prove: "for all $\varepsilon > 0$, we have $(\varepsilon + 1)^2 = (\varepsilon + 1)^2$." (Of course the condition that ε be positive is mathematically superfluous, and is only here for pedagogical purposes.)

If you're thinking that `rfl` will do the trick, that's a good idea, but it won't work, because the goal isn't (yet) an equality. So we need to do something else first.

In Lean, as in mathematics, "for all" is written using ∀; this is called the *universal quantifier*, and is gotten by typing `\forall`, that is, backslash, then `forall`, then a space. In Lean, this goal looks like so:

∀ ε > 0, (ε+ 1)^2 = (ε+ 1)^2.

(Note that to write an epsilon in Lean, you just type `\e`, that is, backslash, then `e`, then space.)

To prove a "for all" statement, you need to show that it's true for an arbitrary element. In English, you would say: give me an arbitrary ε, and give me the fact that it's positive (we can give that fact a name, like `hε`, since it's a hypothesis about ε, or perhaps an even more descriptive name like `ε_pos`, since the hypothesis is the positivity of ε). Note that ε here is a dummy variable, and we could choose to name it something else on the fly. In English, we might say: give me some ε, but I want to call it `Alice`; then give me the fact that `Alice` is positive, and my goal will be to prove that `(Alice + 1)^2 = (Alice + 1)^2`. If we were more polite, we might replace "give me" above with "introduce", like: introduce an ε, and introduce the fact, call it `hε`, that ε is positive.

In Lean, the syntax for this is the command `intro`, followed by whatever name you want to give a dummy variable or a hypothesis.

So: when you see a goal that starts with ∀, you can write `intro` to "introduce" the variable. For example:

- `intro ε` introduces the variable ε. But look at the goal state now! It changes to: `ε > 0` →`(ε+ 1)^2 = (ε+ 1)^2`. So we're not done introducing things.

- Then `intro hε` introduces the hypothesis that `ε > 0` (and again, you can call the hypothesis whatever you want; try `intro ε_pos` instead).

After using `intro` twice, the goal will become one that you should know how to solve.

If you want to be really slick, you can combine the two `intro` commands into one: `intro ε hε`. But don't feel obliged.

Remark. The `intro` tactic introduces variables and hypotheses from ∀ statements or implications.

Theorem 7.1. *For all positive real numbers, this algebraic identity holds.*

```
Statement : ∀ ε : ℝ, ε > 0 → (ε + 1)^2 = (ε + 1)^2 := by
  intro ε
  intro h
  rfl
```

Hint: Use `intro ε` to introduce the variable, then `intro hε` to introduce the hypothesis `ε > 0`. Then how do you solve the goal?

7.2 Conclusion

Excellent! You've learned the `intro` tactic for universal statements.

Notice what happened:

1. `intro ε` introduced the arbitrary real number ε
2. `intro hε` introduced the hypothesis `hε : ε > 0`
3. The goal became `(ε + 1)^2 = (ε+ 1)^2`
4. `rfl` solved the goal, by the reflexive property of the equals sign.

You might have noticed something interesting: we used `intro` in two seemingly different ways—first to introduce an "Object" (the real number ε), and second to introduce an "Assumption" or hypothesis (that `ε > 0`). In Lean's underlying logic ("dependent type theory"), there's actually a deep unity here that mathematicians call the *Curry-Howard correspondence*: propositions are "Types", and proofs are "Terms" of those Types. This means that introducing a hypothesis is really just introducing a term of a certain type, just like introducing a variable.

But here's an even more mind-bending perspective: our entire Statement is really a *function*! Its inputs are first an `ε : ℝ`, then a proof that `ε > 0`, and

its output is a proof that `(ε + 1)^2 = (ε+ 1)^2`. When we write `intro ε` and `intro h`, we're literally defining this function by saying "given these inputs, here's how to compute the output." In this view, all of mathematics – from the simplest definitions to proofs of the deepest theorems – is secretly just functions transforming inputs into outputs!

This beautiful connection between logic and computation underlies much of modern proof assistants, though we won't dive into the details in this course – it's perfectly fine if you didn't follow the last two paragraphs! For now, just appreciate that `intro` works uniformly whether you're introducing mathematical objects or logical assumptions, and that every proof you write is secretly a program!

The `intro` tactic is absolutely crucial in real analysis. You'll use it constantly to:

- Handle "for all $\varepsilon > 0$" statements in limit definitions
- Introduce arbitrary points in domain/range proofs
- Work with function definitions

This pattern of `intro` followed by algebraic manipulation is everywhere in analysis!

8 The specialize tactic

8.1 Using universal statements

Now let's learn the flip side of `intro`. You have already learned that:

- if you have ∃ in the goal, you write `use` to provide a specific value. And
- if you have ∀ in the goal, you write `intro` to introduce an arbitrary variable

But what if you have ∀ in a *hypothesis* and you want to use it for a particular value?

For a concrete example, suppose you have:

- A positive real number `t`; that is, a real number `t`, together with a hypothesis, say, `t_pos` that `t > 0`
- A function `f : ℝ→ℝ`
- A hypothesis `hf : ∀x > 0, f (x)= x^2`, meaning "for all x positive, f (x) equals x^2". (Note that you *have* to put a space after `f` before `(x)` or else Lean will be very angry with you! In fact, Lean will often drop unnecessary parentheses, so you'll see `f x` instead of `f (x)` – and again, definitely *not* `f(x)`.)
- And you want to prove the goal `f (t) = t^2`.

Can you use `exact hf`? No! The hypothesis `hf` says "for all positive x, $f(x) = x^2$" but the goal asks specifically about `f (t) = t ^ 2`. They're not *exactly* the same.

This is where the `specialize` command comes in. You can write `specialize hf t` to specialize the statement `hf` to the particular value `t`. This transforms `hf` from "`∀ x > 0, f (x)= x ^ 2`" into "`t > 0 →f (t)= t ^ 2`". Just like we had to `intro` multiple times (once for the dummy variable name, and again to name the hypothesis), we can specialize multiple times; so you can now write `specialize hf t_pos`. Or you can kill two birds with one stone via: `specialize hf t t_pos`.

I'm sure you can solve the goal from there yourself!

Remark. The `specialize` tactic applies a universal statement in a hypothesis to a specific value.

Theorem 8.1. *If a function of x always equals x^2, then it equals t^2 when evaluated at t.*

```
Statement (t : ℝ) (t_pos : t > 0) (f : ℝ → ℝ) (hf : ∀ x
    > 0, f (x) = x^2) : f (t) = t^2 := by
  specialize hf t
  specialize hf t_pos
  exact hf
```

Hint: Write `specialize hf t` to apply the universal statement to the specific value t.

8.2 Conclusion

Great! You've learned the `specialize` tactic.

Notice what happened:

1. Initially, `hf : ∀x > 0, f (x)= x^2` was a universal statement
2. `specialize hf t` transformed it into `hf : t > 0 →f (t)= t ^ 2`
3. Another `specialize` command, namely `specialize hf t_pos` turned the hypothesis `hf` into `hf : f (t)= t ^ 2`
4. And finally, `exact hf` worked because the hypothesis exactly matched the goal.

The pattern is:

- `intro` when you have ∀ in the goal ("introduce an arbitrary term...")
- `specialize` when you have ∀ in a hypothesis ("apply the hypothesis to specific value...")

This is fundamental in real analysis when working with:

- Function properties that hold "for all x"
- Limit definitions involving "for all $\varepsilon > 0$"
- Continuity statements

Last lesson in Lecture 1 coming up.

9 Big Boss: The Ultimate Tactic Challenge

9.1 The Final Challenge

Congratulations! You've learned many fundamental tactics for mathematical reasoning in Lean:

- `exact hypothesisName` for when a hypothesis matches the goal exactly
- `rfl` for reflexivity (proving `X = X`)
- `rewrite [hypothesisName]` for rewriting using equalities
- `ring_nf` for algebraic manipulation
- `use` for providing witnesses to existence statements in goals
- `intro` for handling universal quantifiers in goals
- `specialize` for applying universal statements to specific values in hypotheses
- `choose value hypothesisOnValue using ExistentialHypothesis` for extracting information from existence statements in hypotheses

Here's a little "Universal/Existential Quantifier Cheat Sheet":

starts with:	∀	∃
Goal	`intro`	`use`
Hypothesis	`specialize`	`choose`

Now it's time for your first **Big Boss** - a problem that requires you to use almost ALL of these tactics in a single proof!

World 1 Big Boss Given a function `f` and information about its behavior, prove a complex statement that involves existence, universals, algebra, and rewriting.

This is what real mathematical proofs look like - a careful orchestration of multiple reasoning steps. You've got this! Use everything you've learned.

Extra Challenge If you want to *really* challenge yourself, play this level "blind". Write the assumptions and goal down on paper, close the computer, solve it by hand, keeping track *in your mind* of what happens to the game

board after each command, and only once you've worked it all out, open the computer and see if Lean accepts your solution.

Why do you think that this would this be a good thing to do?

In general, I hope your *goal* in taking this course is to make your "Real Analysis Brain Muscles" stronger. By the end, you should be *really good* at solving Real Analysis problems on paper, where you don't have Lean showing you the Goal State after every move. More broadly, the purpose of learning to solve Real Analysis problems is to learn to *think*, clearly, precisely. Strengthening your ability to work with pen and paper (or just mentally) directly transfers to *any* other context where you're exploring ideas, wrestling with complicated arguments, or trying to communicate clearly to others.

An LLM could easily work through all these Lean levels by pattern matching and logical manipulation - just as you could solve multiplication problems by plugging them into a calculator instead of memorizing your times tables. But that completely defeats the purpose of the exercise, which is to rewire your brain and build mathematical intuition. It's like deciding that you want to bench press 200 pounds, loading up the bar, and then using a forklift to lift it for you while you stand underneath - you might have moved the weight, but you haven't gotten any stronger. The real value isn't in producing correct proofs, it's in the cognitive transformation that happens when you *struggle* through the reasoning yourself, building the mental pathways that let you see mathematical structure intuitively.

Remark. **BIG BOSS LEVEL**: This problem requires all the tactics you've learned!

Theorem 9.1. *Given a function with specific properties, we can prove a complex statement involving existence and universals.*

```
Statement (f : ℝ → ℝ) (h_existential : ∃ (a : ℝ), f (a)
    = 3)
  (h_universal : ∀ x > 0, f (x + 1) = f (x) + 9) :
  ∃ (b : ℝ), ∀ y > 0, f (y + 1)^2 = (f (y) + (f b)^2)^2
     := by
  -- Step 1: Extract the witness from the existence
     hypothesis
  choose a ha using h_existential
  -- Step 2: We'll use a as our witness for b
  use a
```

```
-- Step 3: Introduce the universal quantifier
intro y
intro hy
-- Step 4: Apply the universal property to our
   specific value a
specialize h_universal y hy
-- Step 5: Rewrite using what we learned about f(y +
   1)
rewrite [h_universal]
-- Step 6: Rewrite using what we know about f(a)
rewrite [ha]
-- Step 7: Simplify the algebra
ring_nf
```

9.2 Conclusion

VICTORY!

You've defeated the Big Boss and mastered all the fundamental tactics of mathematical reasoning!

Let's see what you just accomplished:

1. `choose a ha using h_existential` - Extracted the witness `a` and fact that `f (a) = 3` from the hypothesis
2. `use a` - Chose `a` as your witness for the existence statement in the goal
3. `intro y hy` - Handled the universal quantifier "for all $y > 0$" in the goal
4. `specialize h_universal y hy` - Applied the universal property to your specific value in the hypothesis
5. `rewrite [h_universal]` - Used the specialized fact to rewrite the goal
6. `rewrite [ha]` - Used the original fact that `f (a) = 3` to also rewrite the goal
7. `ring_nf` - Verified finally that `(f y + 9)^ 2 = (f y + 3 ^ 2)^ 2`

You've just completed a genuinely sophisticated mathematical argument! This kind of multi-step reasoning, combining existence statements, universal properties, and algebraic manipulation, is exactly what you'll encounter throughout real analysis.

You are now ready to begin your journey to rigorous calculus!

Welcome to the Introduction to Formal Real Analysis.

9.3 Epilogue

Before we continue with more Real Analysis and more Lean, let's pause to note a few interesting things about working formally. Using a theorem prover interactively is (I hope) tremendously fun and (I hope) leads to rapid gains, immediate feedback, and clarity of thought.

Imagine trying to learn chess by just reading through algebraic notation - 1.e4 e5 2.Nf3 Nf6 3.Bb5 a6 - sure, all the information is technically there, but isn't it so much easier to learn by actually looking at a chess board and seeing how the position changes after each move? In mathematics, it would be extraordinarily tedious to manually write on the blackboard the entire goal state after every move, keeping track of all the hypotheses and their relationships by hand. A theorem prover does this bookkeeping for you automatically, letting you focus on the mathematical content rather than the clerical work.

But! This is, as we've already noted, a double-edged sword. We still want to train our brains to "see" a mental model of the goal state evolving - good chess players can visualize many moves ahead precisely because they've learned to maintain multiple mental game boards simultaneously. But until you develop that skill, and even after you have it, there's immense value in being able to instantly generate the current "game board" of your proof state. The immediate feedback helps you understand the consequences of each logical move, building the very intuition that will eventually let you work more independently. It's the difference between learning to navigate by always checking your GPS versus eventually developing an internal sense of direction - both have their place, and the former helps develop the latter.