Lean for Scientists and Engineers

Tyler R. Josephson

AI & Theory-Oriented Molecular Science (ATOMS) Lab

University of Maryland, Baltimore County

Boba Beach
Grynpyret

Twitter: @trjosephson
Email: tjo@umbc.edu

Lean for Scientists and Engineers 2024

1. Logic and proofs for scientists and engineers
    1. Introduction to theorem proving
    2. Writing proofs in Lean
    3. Formalizing derivations in science and engineering
2. Functional programming in Lean 4
    1. Functional vs. imperative programming
    2. Numerical vs. symbolic mathematics
    3. Writing executable programs in Lean
3. Provably-correct programs for scientific computing

Schedule (tentative)

Logic and proofs for scientists and engineers
Functional programming in Lean 4
Provably-correct programs for scientific computing

July 9, 2024	Introduction to Lean and proofs
July 10, 2024	Equalities and inequalities
July 16, 2024	Proofs with structure
July 17, 2024	Proofs with structure II
July 23, 2024	Proofs about functions; types
July 24, 2024	Calculus-based-proofs
July 30-31, 2024	Prof. Josephson traveling
August 6, 2024	Functions, definitions, structures, recursion
August 8, 2024	Polymorphic functions for floats and reals, compiling Lean to C
August 13, 2024	Input / output, lists, arrays, and indexing
August 14, 2024	Lists, arrays, indexing, and matrices
August 20, 2024	LeanMD & BET Analysis in Lean
August 21, 2024	SciLean tutorial, by Tomáš Skřivan

Content inspired by:
Mechanics of Proof, by Heather Macbeth
Functional Programming in Lean, by David Christiansen

Guest instructor: Tomáš Skřivan

Schedule for today

1. Recap Lecture 5
2. More on function types
3. “Junk” values
4. Calculus in Lean

Slide from Lecture 3

How to find tactics

- Keep learning them one by one!
- Indexes for Mechanics of Proof, Mathematics in Lean
- Consult lists of useful tactics
  - https://github.com/madvorak/lean4-tactics
  - https://github.com/Colin166/Lean4/blob/main/UsefulTactics
- If you have a tactic in hand, mouseover in VS Code to see documentation and example(s)

Slide from Lecture 3

How to find theorems

- Keep practicing!
- Search Mathlib documentation
  - https://leanprover-community.github.io/mathlib4_docs/
  - Using the search bar, make a guess about what the theorem would be named, and start checking things that look promising
- Moogle
  - https://www.moogle.ai
  - Describe theorem (or definition) in natural language, the scroll through options
- Consult lists of useful theorems
  - https://github.com/Colin166/Lean4/blob/main/UsefulLemmas.lean
- If you have a theorem in hand, mouseover in VS Code to see documentation and example(s)

Slide from Lecture 3

Glossary of logical symbols

$\wedge$ - and

$\vee$ - or

$\neg$ - not

$\rightarrow$ - implies

$\leftrightarrow$ - if and only if (implies in both directions)

$\exists$ - exists

$\forall$ - for all

Slide from Lecture 5

Functions: Programming vs. Math

Programming perspective

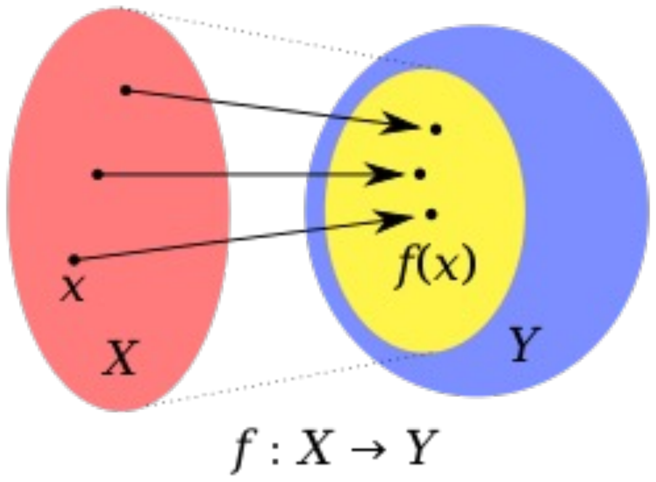
A function takes arguments, performs calculations, and produces an output

Examples in Python

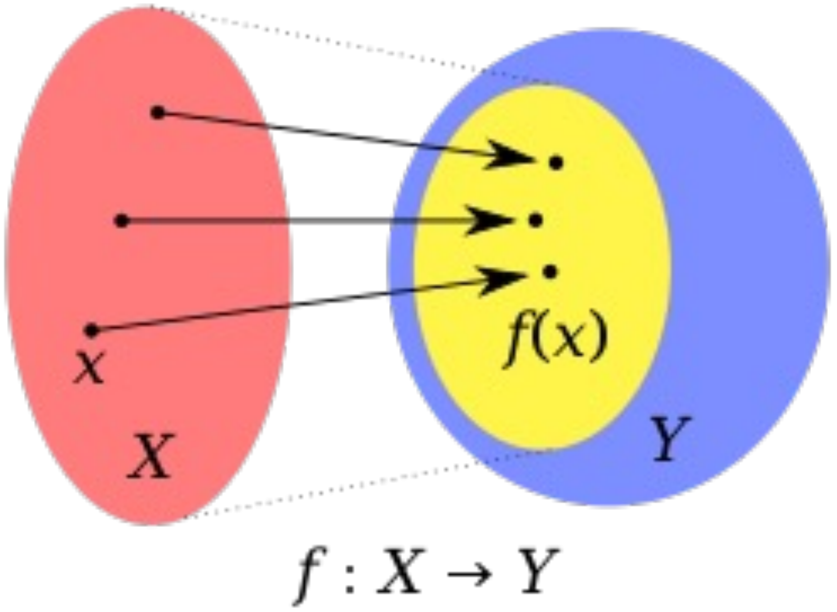
```
def squared(x):
    y = x*x
    return y
```

Math perspective

A function maps values from a domain to a co-domain



Functions: Programming vs. Math



Domain

Co-domain

Image

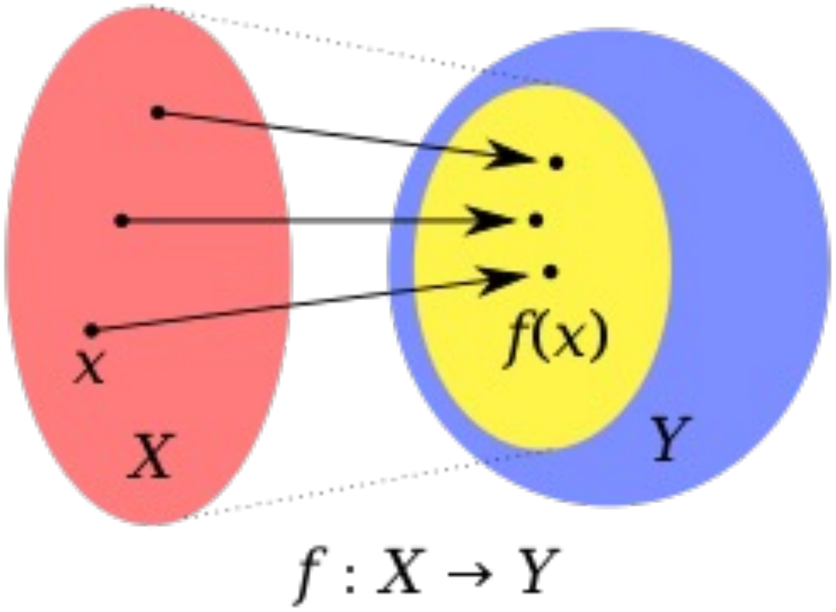
```
def squareroot(x):
    y = x**(1/2)
    return y
```

$$f(x) = \sqrt{x}$$

Not always a function!
With type $\mathbb{Z} \rightarrow \mathbb{Z}$ or $\mathbb{R} \rightarrow \mathbb{R}$, there is no mapping from the $x < 0$ part of the domain

With type $\mathbb{N} \rightarrow \mathbb{R}$ or $\mathbb{R} \rightarrow \mathbb{C}$, it *is* a function; every part of the domain maps to a value in the co-domain

Functions: Programming vs. Math




```
def division(x,y):
    z = x/y
    return z
```

$$f(x,y) = \frac{x}{y}$$

Not a function!

Type is $\mathbb{R} \to \mathbb{R} \to \mathbb{R}$

Everyplace in the domain maps to an place in the codomain, *except* for y = 0

So… what do you do?

$$y \neq 0 \qquad f(x,y) = \frac{x}{y}$$

$$y = 0 \qquad f(x,y) = 0$$

Domain

Co-domain

Image

https://xenaproject.wordpress.com/2020/07/05/division-by-zero-in-type-theory-a-faq/

Slide from Lecture 1

A guide to number systems

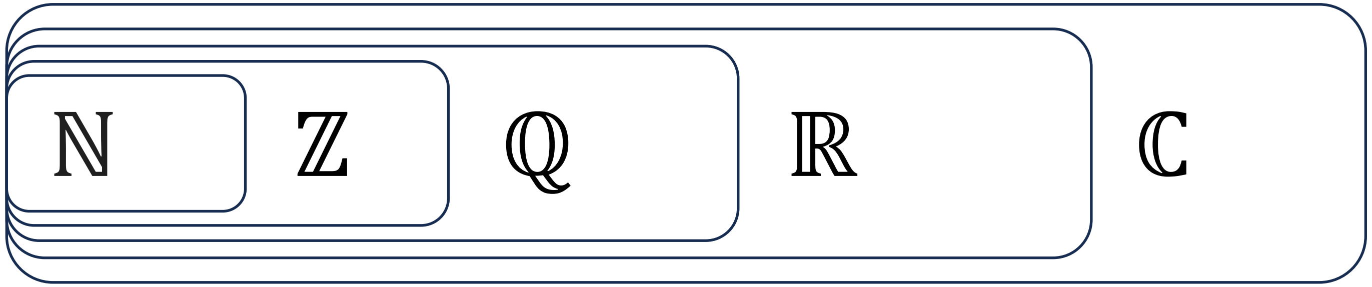
$\mathbb{N}$ - Natural numbers (0, 1, 2, 3, 4, …)

$\mathbb{Z}$ - Integers (… -3, -2, -1, 0, 1, 2, …)

$\mathbb{Q}$ - Rational numbers (1/2, 3/4, 5/9, etc.)

$\mathbb{R}$ - Real numbers (-1, 3.6, π, √2)

$\mathbb{C}$ - Complex numbers (-1, 5 + 2i, √2 + 5i, etc.)



Slide from Lecture 5

Examples of functions

What's a good type?

$I(t)$	Electric current as a function of time
$T(x, y)$	Temperature as a function of position, Cartesian coordinates
$T(r, \theta)$	Temperature as a function of position, polar coordinates
P_n	Pressure as a function of thermodynamic state
$\delta(x)$	Detector threshold as a function of measurement

Currying

$$f(x) = x^2$$

Type $\mathbb{N} \rightarrow \mathbb{N}$

$$f(x, y) = x * y$$

Type $\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$
All binary operators do this

This is called "currying" – a function with multiple arguments is transformed into a series of functions with single arguments

Nat.mul : $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$
Nat.mul x : $\mathbb{N} \rightarrow \mathbb{N}$
Nat.mul x y : $\mathbb{N}$

Junk Values

Junk values

For ℕ 1/0 = 0

For ℕ, 1-2 = 0

Real.sqrt(-5) = 0

Weird, but it makes sense if you think about number systems

For ℕ 4/3 = 1

For ℕ, 2^(1/2) = 1

Nat.sqrt(8) = 2

deriv

- deriv f : $\mathbb{R} \to \mathbb{R}$ is a function that returns the derivative *if it exists* and returns 0 otherwise
  - You provide the function f and tell it what you're taking the derivative of, and you get
- If the derivative exists (i.e., ∃ f', HasDerivAt f f' x),
  then f x' = f x + (x' - x) • deriv f x + o(x' - x) where x' converges to x.
  - This is about *filters* - a generalization of limits
  - Learn more in Topology chapter of Mathematics in Lean
- Notice the type, this maps $\mathbb{R}$ as input to $\mathbb{R}$ as output
  - Deriv does not map a *function* to a *function*, it maps a *function* and a $\mathbb{R}$ to a $\mathbb{R}$

Examples! (showing off some tactics today)

- simp? and aesop? are helpful tactics that find lemmas in Mathlib to solve your problem
- simp and aesop do the same thing, but don't show you what lemmas were found
- The show tactic is like the have tactic, but it's used in-place so you don't even generate a hypothesis

The product rule

$$(u \cdot v)' = u' \cdot v + u \cdot v'$$

In Lean, access this with
rw [deriv_mul]

This creates 3 goals:

1) Prove the calculation above is correct
2) Prove that u’ is differentiable
3) Prove that v’ is differentiable