Intro to Haskell

SIG Math Fall 2025

About today's presentation

- Today's presentation is going to be structured differently. The slides will be very short, but most of the presentation will be a live coding experience in Haskell!
- I recommend you all follow along with the live coding snippets as it will get you in the Haskell mindset and help you retain the information. It's entirely optional – if you are getting confused it's totally fine to just relax and watch the presentation.

Haskell vs other programming languages

- Most programming languages are **imperative** and based on **destructive updates** – you write a series of commands that change state.
- Haskell is **fully functional:**
  - All functions are pure and have no side effects
  - Functions are first-class citizens
  - There is a very rich type system that lets all of this work
  - All of the values in Haskell are lazily evaluated
- As a result, programming in Haskell can feel much more like math, and it's preferred by many mathematicians!
- Sidenote: I dislike the term "functional programming" because it seems to imply "programming with functions", which everyone does! Functional programming implies a lot more than that, which you'll see soon.

Functions in Haskell

- Functions in Haskell have two key properties: they are **first-class citizens**, and they are **pure**.
- First-class citizens:
  - **Functions are treated just like any other object in the language**, such as numbers or strings. For example, they can be passed as function arguments or assigned to variables.
  - In C or Java, you cannot write a function that takes another function as an argument. In Haskell or JavaScript, you can.
- Pure:
  - A pure function is one that is:
    - **Deterministic**: always gives the same output for the same input
    - **Side effect-free**: no side effects (printing, modifying global variables, crashing, etc)
  - Haskell does have ways to write impure functions, such as random() or print(). These require special structures known as *monads*.

Running programs

- When running a C program, there are typically a few steps:
  - Ensure your program has a int main(){} function
  - Compile your code using gcc file.c
  - Run your code using ./file
- Haskell works similarly! To compile and run a Haskell program:
  - Ensure your program has a main :: IO () function
  - Compile your code using ghc file.hs
  - Run your code using ./file
- A lot of times compiling code can take a while and is not good for testing or experimentation. So, we will be using the Haskell REPL called **ghci**. ghci lets us run our functions, inspect their types and values, and much more. To enter ghci, run ghci file.hs

Let’s get coding!

Unpacking square

- Programming has a lot of numeric types. Int, Float, Double, Natural, all the differently sized Ints, etc.
- Suppose you want to write a function that squares a number in C. You'd need to implement a squareInt, squareFloat, squareDouble, etc, though the function body would always just be `return a * a;`
- Let's try to think about it from a type perspective. When you **square a value of a certain type, you get that type back**. So our function should probably look something like `a -> a`.
- But not all types can be squared! You can't square a string. In particular, you can only square things that you can multiply (like elements of a ring). We can add a restriction on our a type by writing `Num a => a -> a`.

Haskell vs Purescript

- PureScript is very similar to Haskell, the primary difference being that it compiles to JavaScript instead of machine code.
  - This enables it to be used for website making!
- The difference between PureScript and Haskell largely comes down to developer experience, but there are some small language differences:
  - Num is Ring, Integral is EuclideanRing, and Fractional is DivisionRing, which makes the differences between the numeric types easier to remember if you know some math.
  - PureScript is strict by default while Haskell is lazy. Some people prefer it being strict since that is more natural, but you get a higher risk of stack overflows especially with recursion.
  - PureScript has better handling for partial functions (functions that may crash)