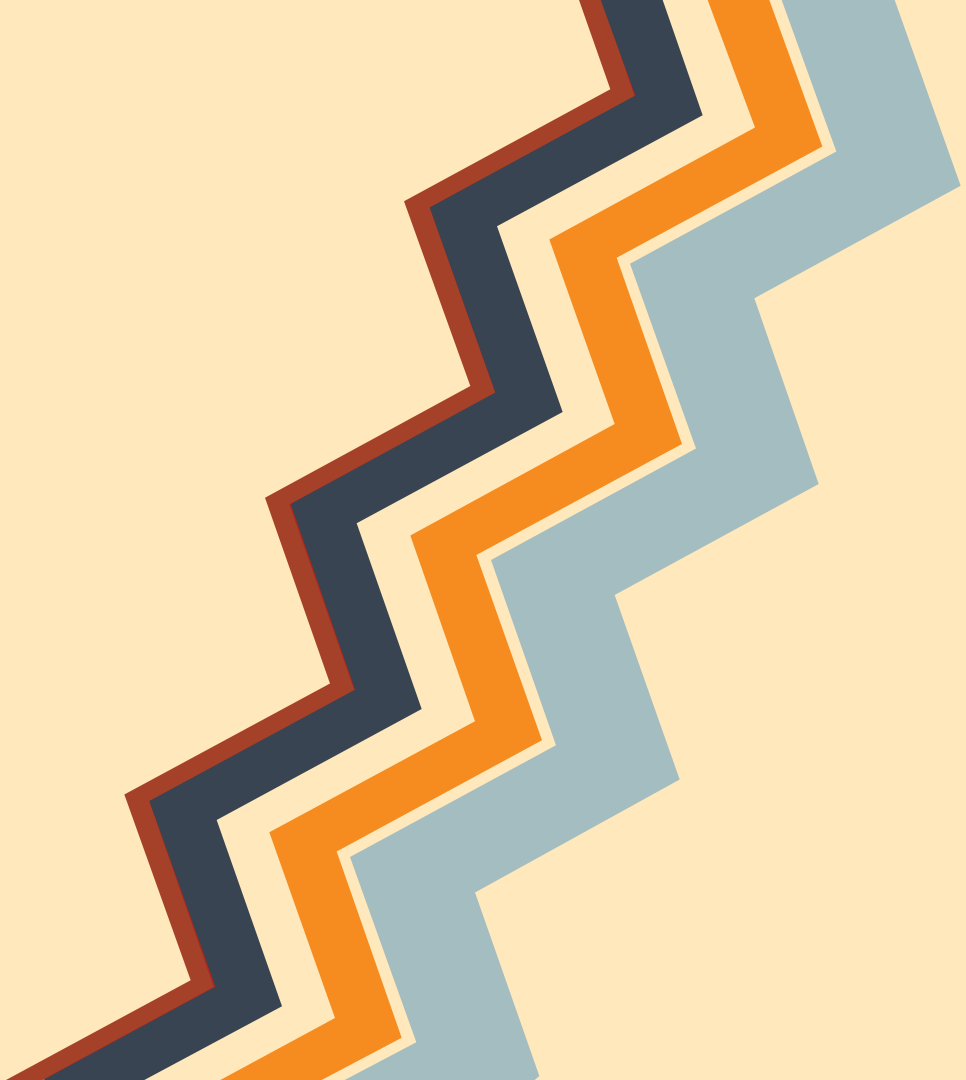
Intro to Functionaλ Programming

by UPλVλS

where λ = “L” | “l”



A dichotomy

- Imperative (how to do)
  - C, C++, Java, JavaScript, Python, Ruby
  - Variables, mutability, and state
  - Assembly-/DIY-mentality
  - Loops and other restrictions => assembly
- Functional (what to do -> **meaning**)
  - Haskell, Clojure, Scheme, OCaml, F#
  - (Mostly) immutable, stateless, (mostly) pure
  - No loops, no variables, only recursion
  - Pattern matching, first-class functions, currying

Example

```java
// Java
IntList copyPosMult (IntList orig, int k) {
	IntList res = new IntList();
	for (int i = 0; i < orig.size(); i++) {
		if (orig[i] != NULL
			&& orig[i].value() > 0) {
			res[i] = orig[i] * k;
		}
		else {
			res[i] = 0;
		}
	}
	return res;
}
```

```haskell
-- Haskell
posMultMap :: Int -> Maybe Int -> Int
posMultMap k (Just n) = if (n > 0) then (k * n) else 0
posMultMap k Nothing = 0

copyPosMult :: [Maybe Int] -> Int -> [Int]
copyPosMult origList k = map (posMultMap k) origList

-- since we finished early, let's write a memoized Fib
fibs  = zipWith (+) ([0,1] ++ fibs) ([1] ++ fibs)
fib n = fibs !! n
```

The Essence of FP

IMMUTABILITY

↓

RECURSION

↓

MATHEMATICAL MODULARITY

Why FP? (opinions)

- Elegance in abstraction
- “Easier to reason about programs”
- Safety in logic, less buggy
- Modular building blocks
- Let the compiler do the work
- Parallel computing and event-based/async programming
- Fun, challenging, and educational

Let’s begin!

- Downloads for Mac, Linux, and Windows
- Why Haskell?
  - Powerful type system
  - “If it compiles, it’s bug-free” -- sort of
  - Easier syntax, but IO & other monads are “tough”
- We’ll use Haskell’s REPL, `ghci`, so IO will be easy regardless; feel free to use `ghc`!
- Start the adventure here