I'll Miss You

Looking back, looking ahead

So you just… learned about programming languages?

Big learning goals

1. Define, analyze, and modify syntactic features of a programming language.

2. Define and analyze semantic features of a programming language.

3. Write programs that operate on other programs.

And along the way…

Expand your definition of “programming”.

Computer languages differ not so much in what they make possible, but in what they make easy.

Larry Wall (creator of Perl)

The Blub Paradox (by Paul Graham)

I'm going to use a hypothetical language called Blub,
and a hypothetical Blub programmer named David.

Blub falls right in the middle of the abstractness continuum. It is not the most powerful language, but it is more powerful than Cobol or machine language.

Of course David wouldn't program in machine language. That's what compilers are for. And as for Cobol, he doesn't know how anyone can get anything done with it. It doesn't even have [Blub feature of your choice].

Languages less powerful than Blub are obviously less powerful, because they're missing some feature David's used to.

But when David looks up the power continuum, what he sees are merely weird languages. They seem about equivalent in power to Blub, but with all this other hairy stuff thrown in.

Blub is good enough for him, because he thinks in Blub.

The Lambda Calculus and functional programming

Pure functions as the basic unit of abstraction;
evaluation by substitution.

```
(+ (* 22 1.8) 32)
(+ (* -7 1.8) 32)
```

```
(lambda (x) (+ (* x 1.8) 32))
```

```
(define (add1ToAll lst)
  (if (empty? lst)
      '()
      (cons ((add1 (first lst))
             (add1ToAll (rest lst))))))

(define (sqrAll lst)
  (if (empty? lst)
      '()
      (cons ((sqr (first lst))
             (sqrAll (rest lst))))))

(map f lst)
```

Sorting as a higher-order function

```
(sort lst less-than?)

list.sort(lst, key)

Collections.sort(lst, Comparator)
```

Name bindings: environments, closures, scope

How are name bindings resolved?
Static vs. dynamic semantics

TemplateHaskell

Compiler extensions

lst[i] -> __getitem__

Extending language syntax

Doing things that functions can't do
(bind new names, circumvent normal evaluation order)

Two large macro libraries

my-class:

a dynamic, message-passing approach to OOP

choice:

multiple possibilities in a single expression

Macros push the limits of expressing what we want to compute.

Type systems push the limits of expressing constraints on our computations.

Polymorphism is a form of abstraction

```
headInts  :: [Int]  -> Int
headBools :: [Bool] -> Bool

head      :: [a]    -> a
```

Polymorphism is a form of abstraction

```
mapList   :: (a -> b) -> [a] -> [b]
mapVector :: (a -> b) -> Vector a -> Vector b

fmap      :: Functor f => (a -> b) -> f a -> f b
```

Monads are an abstraction of chainable computations

Computations that can fail

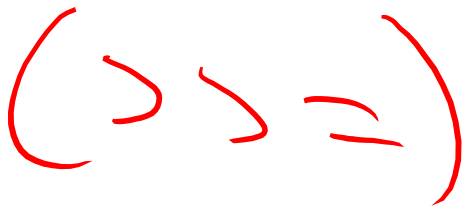
```
Maybe a, Either String a
```

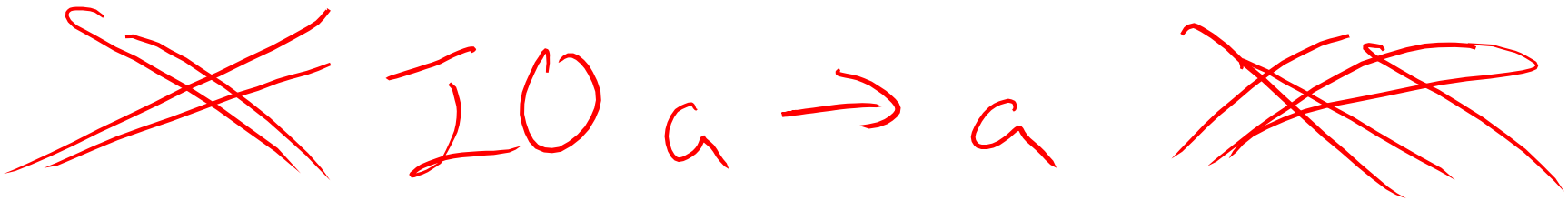
Computations using "mutable" state

```
State s a
```

Computations that perform I/O

```
IO a
```





Haskell's type system expresses *purity by default*

In Haskell, an `Int -> Int` function is *guaranteed* to not do anything except return an `Int`.

It can't return null, raise an error, mutate a global variable, or perform some I/O action.

const cat

And now, a few words on what comes next.

The classic: writing a compiler

In this course, we focused on *dynamic execution of a program*, performed by an interpreter.

Compilers are fundamentally static, and target machine code.

Implementing *zero-cost abstractions*

How can we take many of the abstractions we've discussed in this course, and implement them efficiently?

"Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety."

www.rust-lang.org

Formal verification: static proofs of correctness

How can we specify a program's behaviour, and automatically prove its correctness without running a single test?

Dependent types: mixing values and types

```
[1, 2, 3] :: Vector Int 3
```

Thank you for being a great class!

Good luck on the final exam,
and have a restful holiday break!