

CS 252:

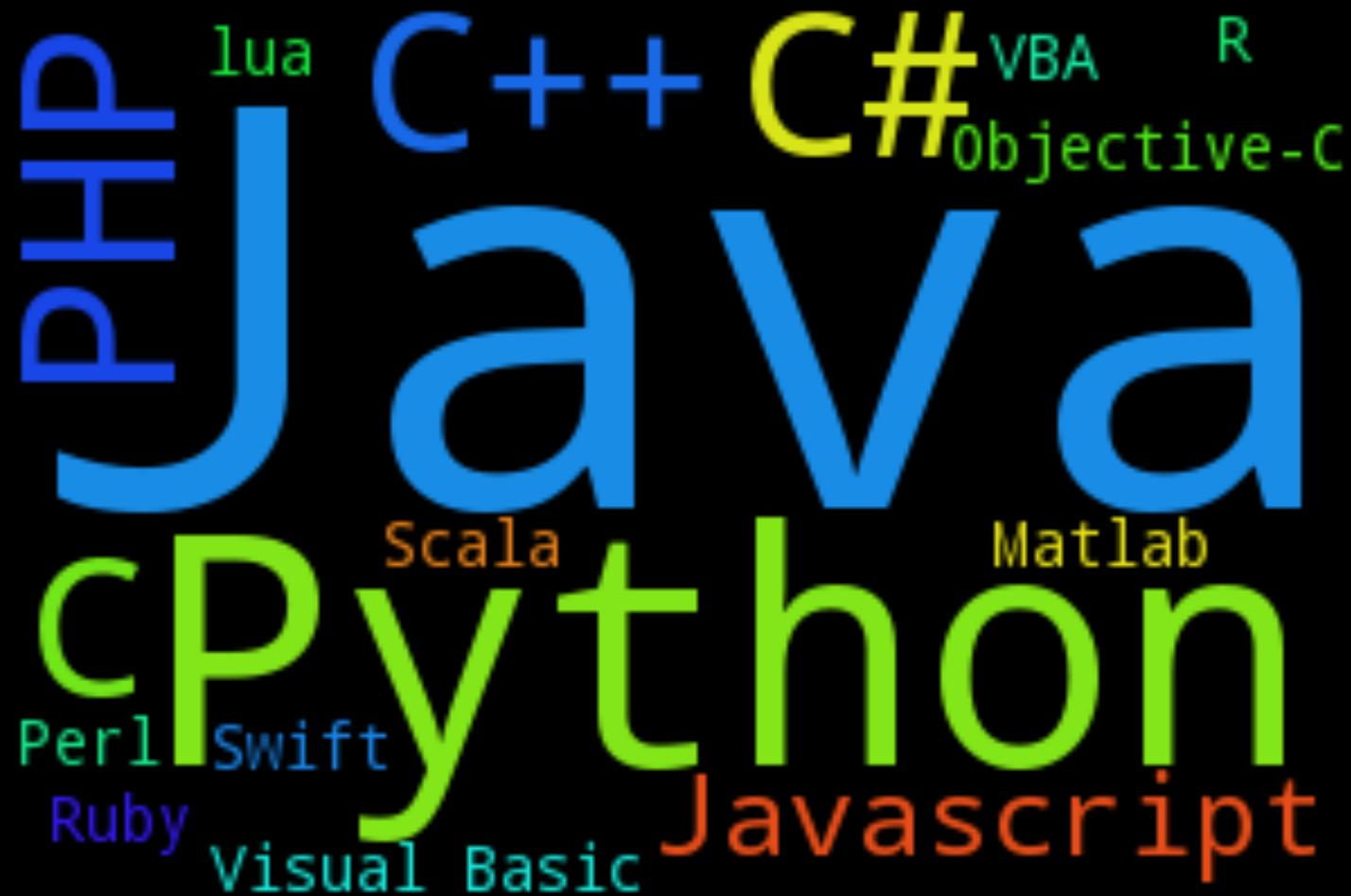
Advanced Programming Language Principles



Prof. Tom Austin

San José State University

What are some
programming languages?

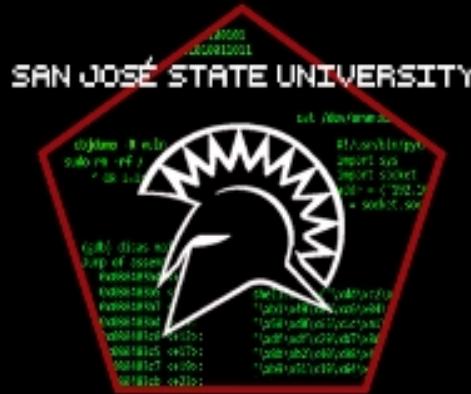


Taken from <http://pypl.github.io/PYPL.html>

January 2016

Why are there so many?

Different domains



INFOSEC CLUB



Different design choices

- Flexibility
- Type safety
- Performance
- Build time
- Concurrency

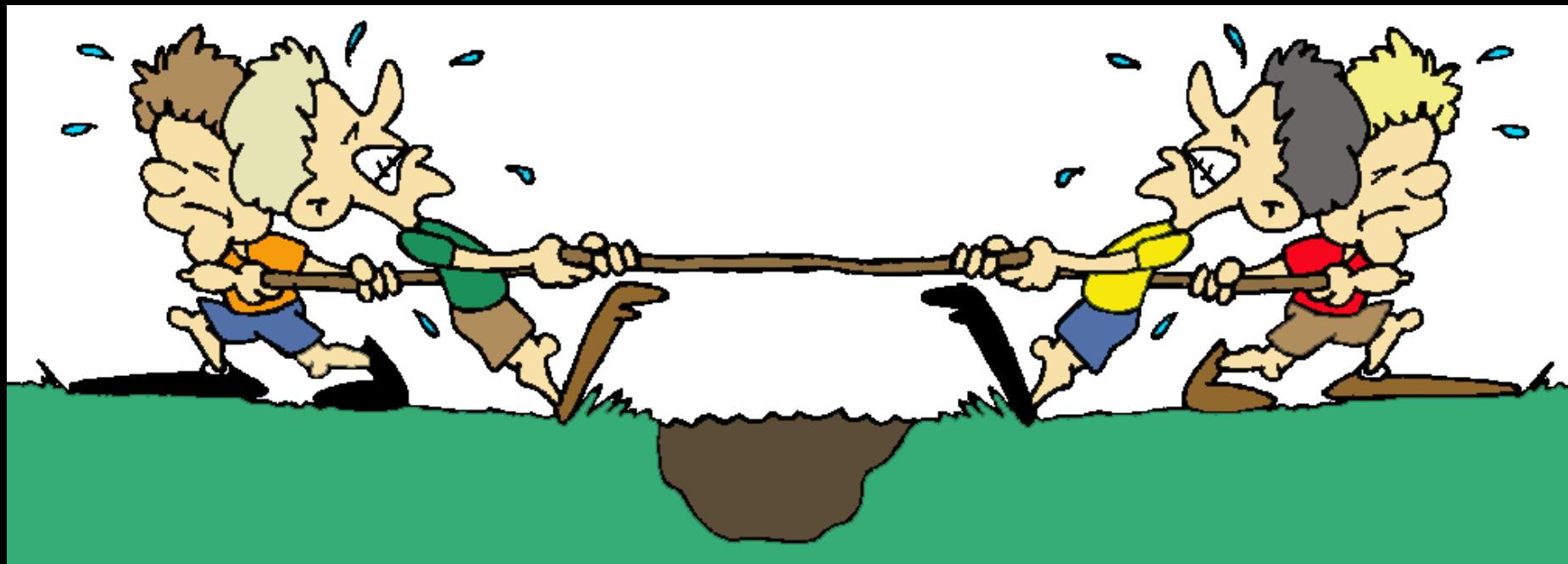


Which language is better?

Good language features

- Simplicity
- Readability
- Learn-ability
- Safety
- Machine independence
- Efficiency

These goals almost always conflict



Conflict: Type Systems

Stop "bad" programs

... but ...

restrict the programmer

Why do we study
programming languages?

For undergrads:
to warp their minds

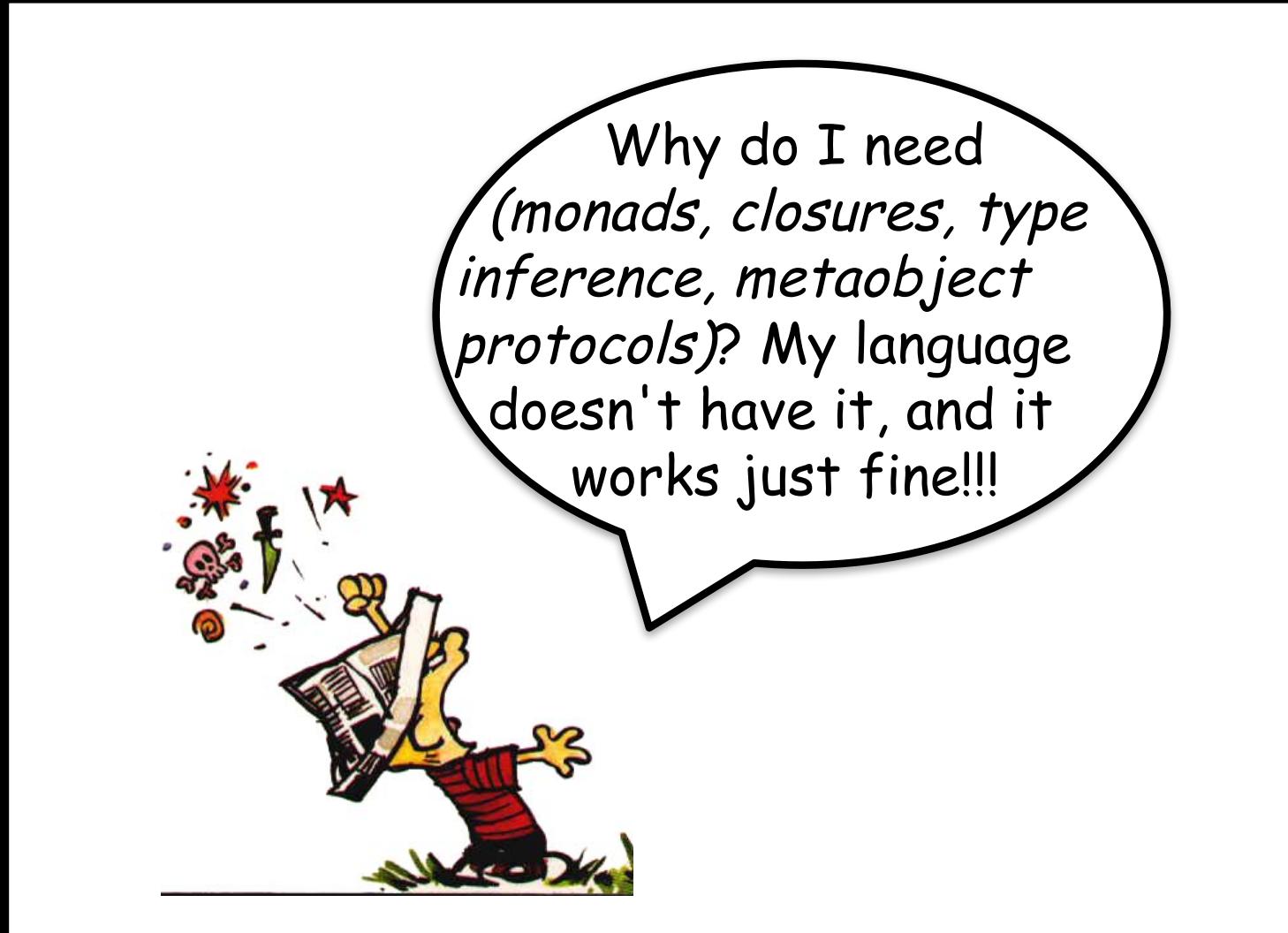


Objectives for grad students:



- Understand advanced language features
- *Evaluate* different features
- Choose the right language
- Understand formalisms

The "Blub" paradox



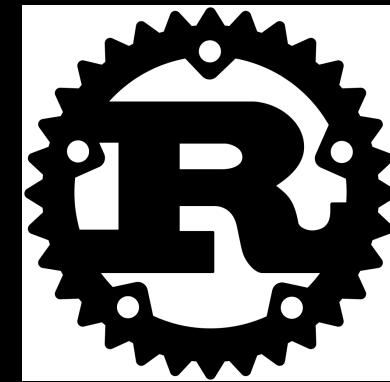
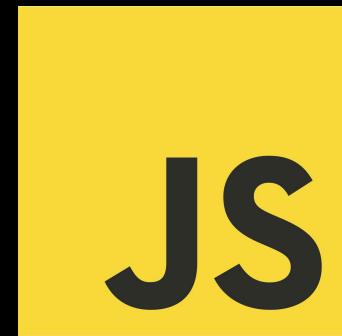
"As long as our hypothetical Blub programmer is looking down the power continuum, he knows he's looking down..."

[Blub programmers are] satisfied with whatever language they happen to use, because it dictates the way they think about programs."

--Paul Graham

<http://www.paulgraham.com/avg.html>

Languages we will cover (subject to change)



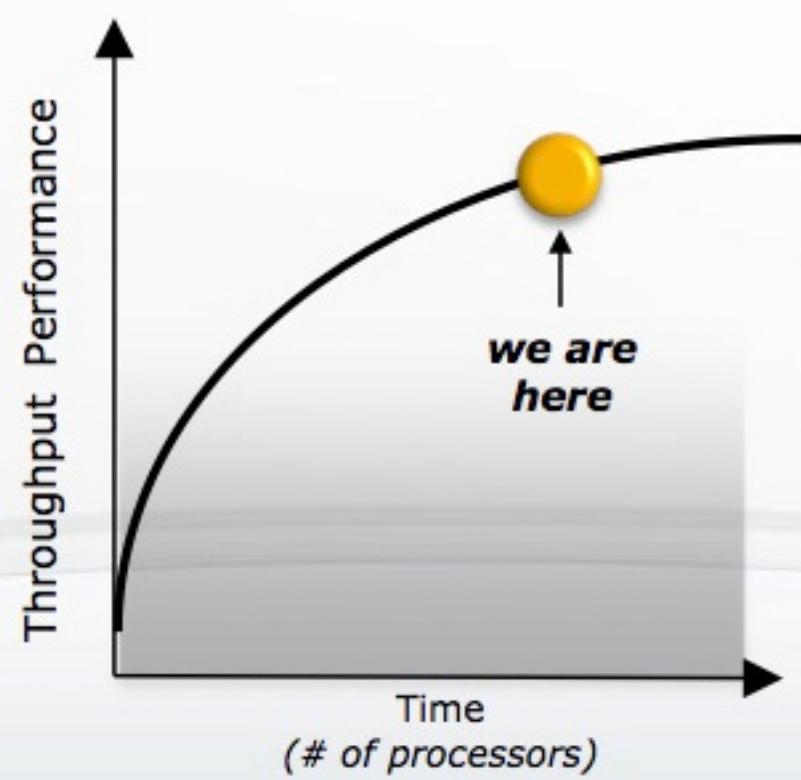
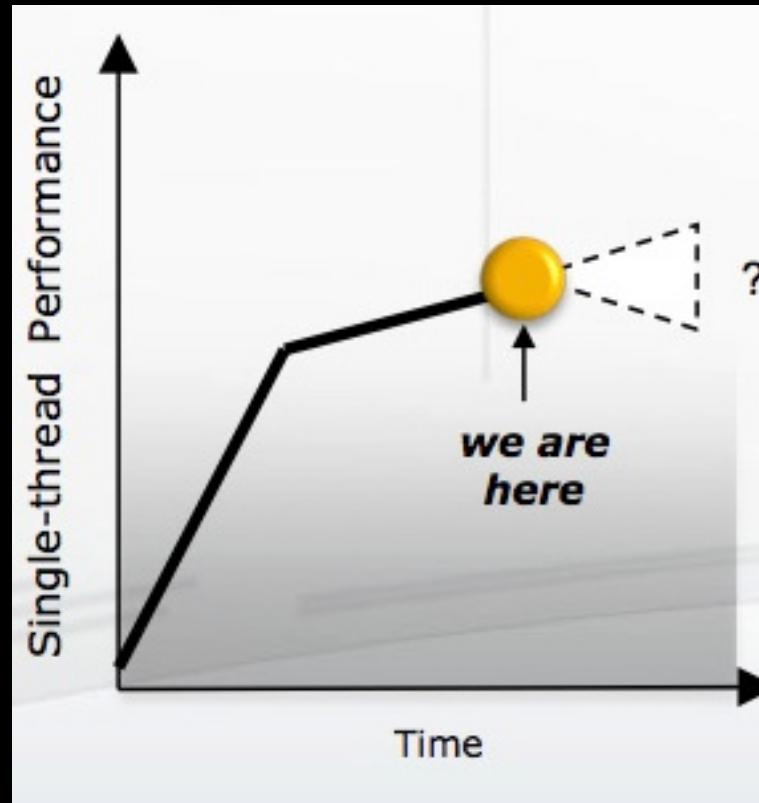
Theoretical foundations topics

- Formal semantics
- Type systems
- Concurrency approaches
- Metaprogramming
- Security features

In this course, you will learn the
practical *and* the theoretical

What are the PL issues of
interest to industry?

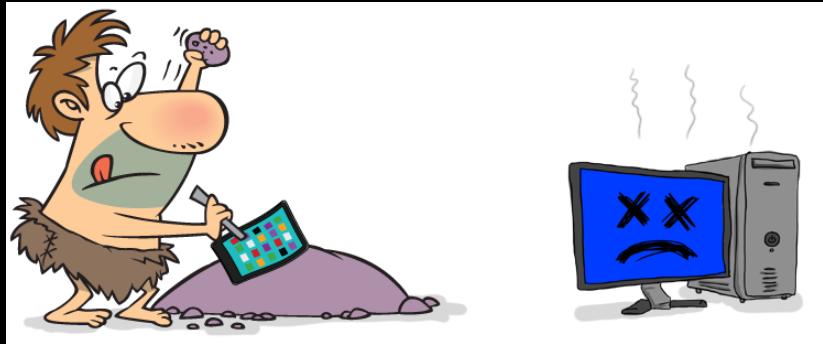
Multi-core explosion



Big Data

A dense word cloud centered around the term "HADOOP". Other prominent words include "BIG DATA", "OPEN SOURCE", "ANALYTICS", "DATA", "CLOUD", "INTERNET", "SOCIAL MEDIA", and "SECURITY". The words are in various sizes and colors (yellow, orange, red) against a dark background.

Mobile Devices



Rise of the Web & Scripting Languages

- Perl, Python, Ruby, PHP, JavaScript...
- Flexible
- Dynamically typed
- Easy to get started
- Minimal typing



Hello world in Java

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Hello world in Ruby

```
puts "Hello World!"
```

What are the PL academics interested in?

- Types
- Types
- More types!

(OK, maybe a *little* more than that...)

Why are they different?

Major PL research contributions

- Garbage collection
- *Sound* type systems
- Concurrency tools
- Closures

But PL researchers have missed
some areas of interest

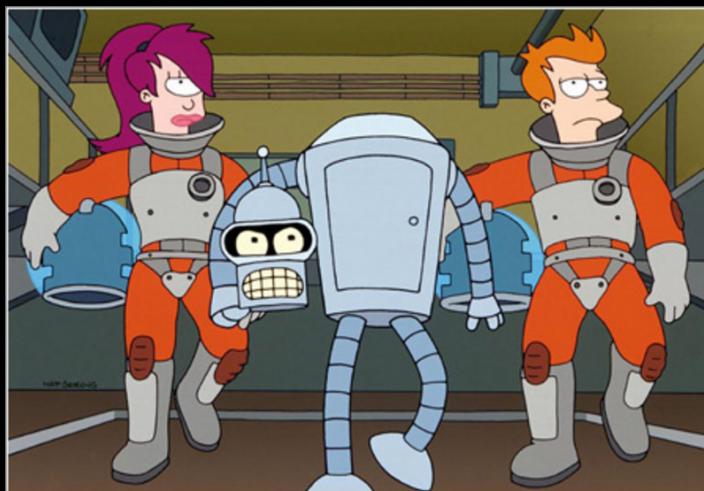
Academia and industry: Idealized relationship



Academics come up with
brilliant ideas



Academia shares its ideas
with people in industry

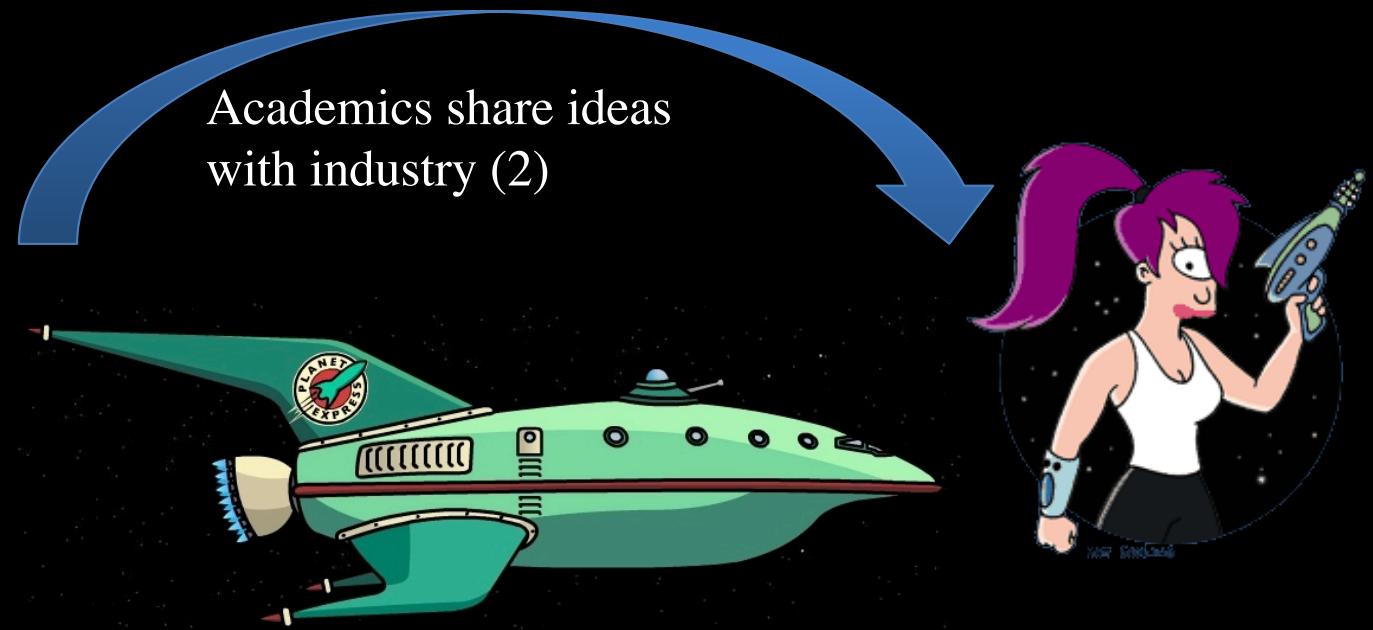


Engineers use these ideas
to build great things

Actual relationship



Academics come up with ideas (1)



Engineers use these ideas, incorporating their own ideas and experiences (3)

Actual relationship



Academics come up with ideas (1)

The cycle continues (6)

Academics share ideas with industry (2)

Sometimes they don't work out well (4)

Academics learn from the experience of engineers (5)



To participate in this discussion,
you must understand
formal semantics.

Programs that manipulate other programs

- compilers & interpreters
- JavaScript rewriting
- instrumentation
- program analyzers
- IDEs

Sharing ideas *unambiguously*

- ECMAScript committee
- Highlight issues early in the design
- Prove that a language supports a given property

Three approaches to semantics

- Operational
 - big-step
 - small-step
- Axiomatic
- Denotational

$$\frac{\Gamma \vdash e \Downarrow \text{true} \quad \Gamma \vdash e_1 \Downarrow v}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v}$$
$$\frac{\Gamma \vdash e \Downarrow \text{false} \quad \Gamma \vdash e_2 \Downarrow v}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v}$$

Now we can determine in some
objective sense whether a
language feature is "good".

Administrative Details

- Green sheet available at
<http://cs.sjsu.edu/~austin/cs252-fall17/greensheet.html>.
- Assignments will be submitted through Canvas
(<https://sjsu.instructure.com/>)
- Academic integrity policy:
<http://info.sjsu.edu/static/catalog/integrity.html>

Schedule

- Greensheet: *tentative* schedule
- Official schedule on Canvas
- Late homeworks will not be accepted
- Check the schedule before every class
- Check the schedule before every class
- And finally, **CHECK THE SCHEDULE BEFORE EVERY CLASS.**

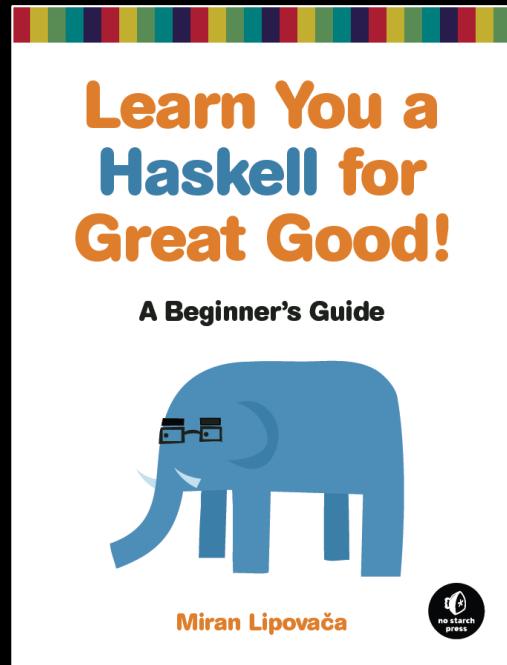
Prerequisites

You should be comfortable with:

- Functional programming
- Mathematical notation

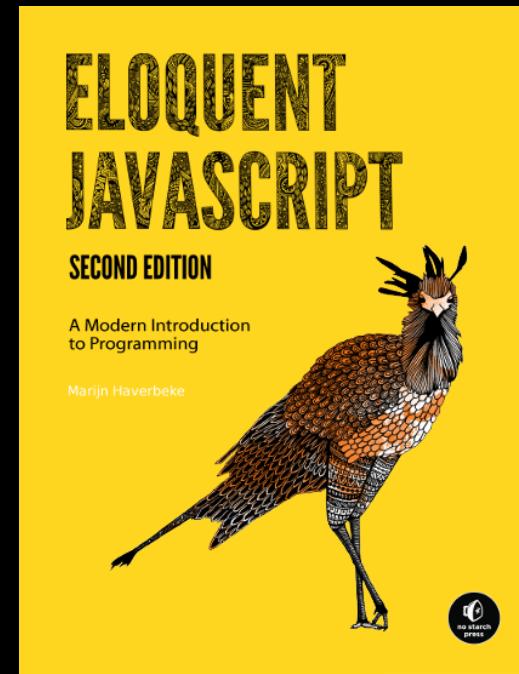
If you are not sure, please see me

Resources



<http://learnyouahaskell.com/>

<http://eloquentjavascript.net>



All available
online!!!

Grading

- 30% -- Homework
- 20% -- Midterm
- 20% -- Final
- 20% -- Project
- 10% -- Labs

Open-ended.

Labs

- Graded complete/incomplete
- I will look at them
- I *might* give feedback
- May show up on exams

Office hours

- MacQuarrie Hall room 216.
- Mondays/Thursdays noon-1pm.
- **No office hours September 7th or 11th.**
- Also available by appointment



Haskell

Haskell is *purely functional*

- We define "what stuff is"
- No side effects
- *Referential transparency*

You can replace an expression with its value and you won't change anything.

Wait, no side effects?!
How is that possible?

Haskell & side effects

- Haskell functions *can* have side effects
 - e.g. file I/O
- BUT, *pure* functions can't call functions with side effects

Haskell supports *type inference*

Some languages have *explicit* types

```
// Java code
String foo(int i) {
    String s = "hello " + i;
    return s;
}
```

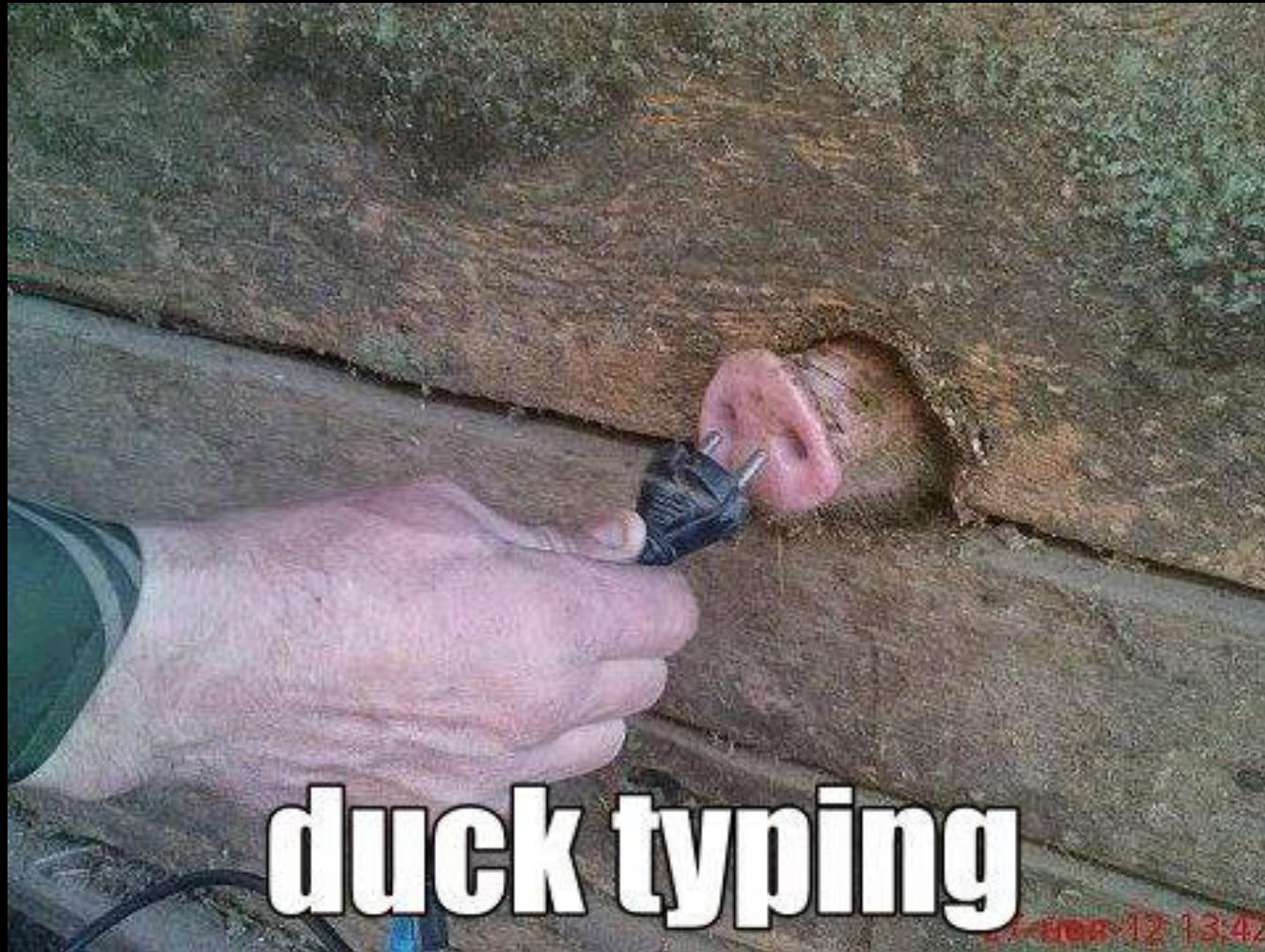
"Scripting languages" use dynamic typing

```
// Ruby code  
def foo(i)  
  s = "hello #{i}"  
  return s;  
end
```



Duck typing

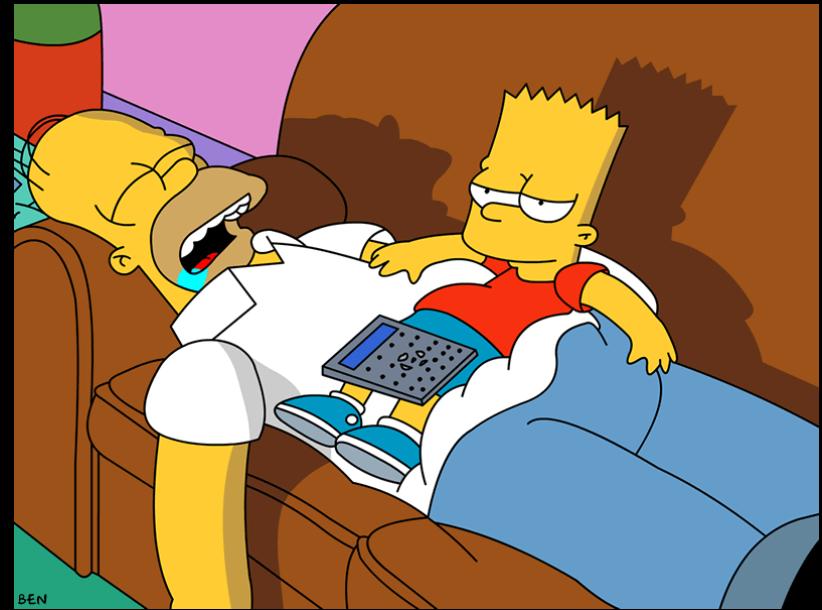
"Duck typing" is flexible but not safe



In Haskell, you do not need
to declare types;
the compiler deduces them

Haskell supports *lazy evaluation*

- Results not calculated until needed.
- Can represent infinite data structures.



Lazy Example

Haskell interactive mode:

```
*Main> let oddNumbers =  
[1,3..]  
  
*Main> take 5 oddNumbers  
[1,3,5,7,9]
```

Before next class

- Install Haskell from
<http://www.haskell.org/platform/contents.html>
- Read chapters 1-3 of "Learn You a Haskell".
- Read C.A.R. Hoare's “Hints on Programming Language Design” and write a short summary of his key points.
<http://i.stanford.edu/pub/cstr/reports/cs/tr/73/403/CS-TR-73-403.pdf>

First homework due September 5th

- Available in Canvas.
 - Alternately, see
<http://www.cs.sjsu.edu/~austin/cs252-fall17/hw/hw1/>
- Get started now!

HW1 Overview

```
public class Test {  
    public void main(String[]  
args) {  
        System.out.println(  
            999999999999999999999999 * 2);  
    } }
```

Won't
compile!

With Haskell, there is no issue:

\$ ghci

• • •

In this assignment, you will build
your own BigNum module.

Haskell support.
Have you tried
converting tabs
to spaces?

