What are some programming languages?
Why are there so many?

• Different domains
  – Mobile devices (Objective C)
  – Web programming (Ruby, Python, PHP)
  – Databases (SQL)

• Different design choices. Some concerns:
  – Flexibility (Scripting languages)
  – Type safety (Java, Scala, Haskell)
  – Performance (C, C++, Go)
  – Build time (Go, Rust, D)
  – Concurrency (Erlang, Go, Rust, D)
Which language is better?
Good language features

• Simplicity
• Readability and writability
• Learn-ability (familiarity helps here)
• Safety
• Machine independence
• Efficiency (of execution, but also of compilation)
• Backwards compatibility
These goals almost always conflict

- Type systems prevent bad programs from running, but also restrict what the programmer can do.
- Machine-independence and efficient low-level operations are difficult to pair.
- Good language designs choosing careful compromises for the right set of features.
Why do we study programming languages?

For undergrads, we want to warp their minds and make them better programmers.
Mind warping is good for grad students as well... but we also want you to:

- be familiar with advanced language features
- know how to *evaluate* different features to choose the right language for the job
- understand the necessary formalisms, so that you can take part in the discussion.
Blub programmers

• Paul Graham introduced the concept of "Blub programmers" in a famous blog post ([http://www.paulgraham.com/avg.html](http://www.paulgraham.com/avg.html)).

• "Blub" is a hypothetical language midway in the "power continuum". (He defines "power" roughly as being able to do more with fewer lines of code.)
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

Why do I need (monads, closures, type inference, metaobject protocols)? My language doesn't have it, and it works just fine!!!
Languages we will cover
(subject to change)

- Haskell (focus will be here)
- Ruby (for metaprogramming)
- Go (Concurrency, and trying to kill C++)
- Scala (Domain Specific Languages)
Theoretical foundations topics

- Formal semantics (especially operational semantics)
- Type systems and proofs
- Concurrency approaches
- Metaprogramming
- Security features
In this course, you will learn the practical \textit{and} the theoretical
What are the issues of interest to industry in the development of new language features?
Multi-core explosion

- Single core CPU performance increases have slowed.
- Performance gains now come from multiple cores.
- How can we take advantage of these extra cores?
  - Automatic parallelization of code
  - Security features (secure multi-execution)
Mobile Devices

- Fierce competition has arisen around mobile devices (Android, iPhone).
- "Mobile-only users".
- What unique challenges are there in this space?
- How can language features help address these issues?
Rise of the Web and Scripting Languages

- Perl, Python, Ruby, PHP, JavaScript...
- Flexible: dynamically typed: the type system does not stop valid program *executions*, even if the complete program has issues.
- Easy to get started.
- (Generally) less typing required
Hello world in Java

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

Hello world in Ruby

```ruby
puts "Hello World!"
```
What are the main areas of interest in Programming Languages (PL) research?

• Types
• Types
• More types!

(OK, maybe a little more than that…)
Why are they different?

• PL research helps create concepts used in the languages of the future
  – Garbage collection
  – Advanced type systems
  – Advanced concurrency constructs

• But it also tends to miss some areas of interest to industry programmers.
Idealized relationship between academia and industry

- Academics come up with brilliant ideas
- Academia shares its ideas with people in industry
- Engineers use these ideas to build great things
Actual relationship between academia and industry

Academics come up with ideas (1)

Academics share ideas with industry (2)

Engineers use these ideas, incorporating their own ideas and experiences (3)
Actual relationship between academia and industry

Academics come up with ideas (1)

Academics share ideas with industry (2)

Sometimes they don't work out well (4)

Academics learn from the experience of engineers (5)

The cycle continues (6)
In order to participate in the discussion, you need to be comfortable with formal semantics.
Who needs formal semantics?
No one, except:

• Those who write programs that manipulate other programs:
  – compilers and interpreters
  – program transformation tools (e.g. JavaScript rewriting)
  – instrumentation tools
  – program analyzers
  – software engineering tools
Who needs formal semantics?
No one, except:

- Those who want to describe (unambiguously) a language feature or program transformation
  - ECMAScript committee
  - Semantics help to frame discussion and highlight some issues early in the process
  - We can formally prove that a language supports a given property

- Those who write critical software
Three Approaches to Language Semantics

• Operational semantics
  – how to evaluate or execute a program
  – useful for implementing a compiler or interpreter

• Axiomatic semantics
  – given a program and a set of starting states and inputs, what is the set of final states after execution
  – does it include “assertion failed”?
  – useful for proving program correctness

• Denotational semantics
  – characterize meaning of each procedure as a mathematical function
  – mathematically heavyweight – we will not cover it in this course.
Once we have the tools for a discussion, we can determine in an \textit{objective} sense whether a language or feature is "good".
Administrative Details

• Green sheet available at
  http://www.sjsu.edu/people/thomas.austin/courses/cs252-fall2013/s1/Greensheet.html

• Homework assignments will be submitted through Canvas (https://sjsu.instructure.com/)

• Academic integrity policy:
  http://info.sjsu.edu/static/catalog/integrity.html
Schedule

• The class schedule is available through 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 are expected to have taken an undergraduate programming languages course
• Functional programming should be familiar
• You should be comfortable with mathematical notation
• If you are not sure, please see me
Resources

Learn You a Haskell for Great Good!
A Beginner’s Guide
Miran Lipovača

http://learnyouahaskell.com/

A Tour of Go,
http://tour.golang.org

All available online!!!
Grading

- 50% -- Homework assignments.
- 40% -- Final project.
- 10% -- Participation.

The project is open-ended: do something you find interesting. We will discuss project ideas more in a later class.
Office hours

• MacQuarrie Hall room 216.
• Wednesdays 10am – 11am.
• Thursdays 10am – 11am.
• If you need to meet with me another time, send me an email.
Haskell
Haskell is purely functional

• We define "what stuff is" rather than how to do it.

• No side effects, allowing for referential transparency.

This means that you can replace an expression with its value and you won't change anything.
Wait, no side effects?! How is that possible?

• Some Haskell functions do have side effects, such as file I/O.
• However, Haskell isolates functions that do have side effects, making it easy to write functions without side effects.
• Java is "object-oriented", but you can still write programs that do not follow OO style; similarly you can write Haskell functions with side-effects.
Type inference

• In most typed languages, you must explicitly declare the type of your data and functions

• In scripting languages, you do not need to specify your types, but you will get runtime errors if your types don't work
  – "Duck typing"
  – Flexible, but not safe

• With Haskell, you do not need to declare types, but the compiler checks them anyway (and you can declare them if you wish)
Lazy evaluation

- An almost unique feature of Haskell is that it is lazy – it does not calculate results until they are needed.
- This feature allows Haskell to represent infinite data structures.
Lazy Example

- Start up Haskell interactive mode and type:
  *Main> let oddNumbers = [1,3..]
- Note that this definition works just fine. However, if you now type in oddNumbers, it will print all odd numbers greater than 1.
- You can use this function to print out however many odd numbers you want:
  *Main> take 5 oddNumbers
  [1,3,5,7,9]
First homework due August 30th

- This assignment is designed to get you up and running with Haskell.
- Available in Canvas.
- Get started now!
Getting started with Haskell

• Install Haskell from http://www.haskell.org/platform/contents.html

• Work on examples from Chapter 2 of "Learn You a Haskell" (LYH from now on). http://learnyouahaskell.com/starting-out

• Ask me if you have questions