# Comparative Programming Languages

CSC 372

Spring 2015

psst...Sign up for Piazza while you're waiting!

## Instructor

William Mitchell (`whm`)

Consultant/contractor doing software development and training of software developers. Lots with Java, C++, C, ActionScript, Ruby, Icon, and more. Linux stuff, too.

Occasionally teach a CS course.  (337, 352, 372, and others)

Adjunct lecturer, not a professor.

Education:

BS CS (North Carolina State University, 1981)

MS CS (University of Arizona, 1984)

Incorrect to say "Dr. Mitchell" or "Professor Mitchell"!

## Topic Sequence

- Functional programming with Haskell
- Imperative and object-oriented programming using dynamic typing with Ruby
- Logic programming with Prolog
- Whatever else in the realm of programming languages that we find interesting and have time for.

Note: We'll cover a selection of elements from the languages, not everything.

## Themes running through the course

Discerning the philosophy of a language and how it's manifested.

Assessing the "mental footprint" of a language.

Acquiring a critical eye for language design.

Learning techniques for teaching ourselves a language.

## Syllabus Highlights

Prerequisites

- CSC 127B or CSC 227
- But, this is a 300-level class!

Piazza

- Our forum
- Sign up if you haven't already!

No Teaching Assistants

## Syllabus, continued

Textbooks...

- No texts are required!
- Lectures, handouts, and Piazza postings might be all you need.
- Syllabus has recommendations for supplementary texts, most of which are on Safari.

## Syllabus, continued

Grading

| | |
|---|---|
| • Assignments | 60% |
| • Pop quizzes | 5% |
| • One mid-term | 13% |
| • Final | 22% |

Ten-point scale: >= 90 is A, etc.  Might go lower.

Original Thoughts

- Half-point on final average for each

## Syllabus, continued

Assignments—things like:

- Coding in the various languages
- Short answer and essay questions
- Diagrams
- One video project

Late assignments are not accepted!

No late days!

But, extensions for situations beyond your control.

## Syllabus, continued

Office Hours:

- I love office hours!
- Open-door policy except before class
- Guaranteed hours posted on Piazza
- In-person is most efficient
- Skype preferred for IM
- `http://join.me` preferred for screen sharing
- OK to call my mobile but don't leave voice mail!
  (Send e-mail instead.)

## NO CHEATING!

Capsule summary:

Don't cheat in my class!

Don't make it easy for anybody else to cheat!

**One strike and you're out!**

For a first offense expect this:

Failing grade for course

Permanent transcript annotation

Disallowance of GRO for failing grade

Recommendation for one semester suspension

A typical first step on the road to ruin is sharing your solutions with your best friend, roommate, etc., who swears to just learn from your work and absolutely not turn it in as their work.

## No asking the world for help!

The material covered in lectures, posted on Piazza, etc. should be all you need to do the assignments.

I challenge you to <u>not</u> search the web for solutions for problems on assignments!

**<u>Posting problem-specific questions on websites, IRC channels, mailing lists, etc. will be considered to be cheating!</u>**

Example: *I'm learning Haskell and trying to write a function that returns True iff the parentheses in a string are properly matched. Any suggestions?*

## My Teaching Philosophy

- I work for you!
- My goal: everybody earns an "A" and averages less than ten hours per week on this course, counting lecture time.
- Effective use of office hours, e-mail, IM, and the telephone can equalize differences in learning speed.
- I should be able to answer every pertinent question about course material.
- My goal is zero defects in slides, assignments, etc.
  Bug Bounty: One assignment point
- Everything I'll expect you to know on exams will be covered in class, on assignments, or on Piazza.

## READ THE SYLLABUS!

## Assignment 0

Assignment 0

- On Piazza
- It's a survey
- Due Tuesday, January 20, 9:30am
- Worth 10 points
- Maybe 10 minutes to complete
- Thanks for doing it!

# Pictures &
# Name memorization

# Basic questions about programming languages

## What is a programming language?

A simple definition:

*A system for describing computation.*

It is generally agreed that in order for a language to be considered a programming language it must be *Turing Complete*.

One way to prove a language is Turing Complete is to use it to implement a *Turing Machine*, a theoretical device capable of performing any algorithmic computation.

Curio: **https://github.com/elitheeli/stupid-machines**

What language is most commonly mis-listed on resumes as a programming language?

## Does it matter what language is used?

The two extremes:

- If you've seen one language you've seen them all. Just pick one and get to work.

- Nothing impacts software development so much as the language being used. We must choose very carefully!

## Why study programming languages?

- Learn new ways to think about computation.
- Learn to see languages from a critical viewpoint.
- Improve basis for choosing languages for a task.
- Add some tools to the “toolbox”.
- Increase ability to design a new language.

Speculate: How many programming languages does the average software developer know?

## How old are programming languages?

Plankalkül 1945
Short Code 1949
FORTRAN 1957
ALGOL 1958
COBOL 1959
LISP 1960
BASIC 1964
PL/I 1965
SNOBOL4 1967
SIMULA 67 1967
Pascal 1971
C 1972
Prolog 1972
Smalltalk 1972
ML 1977
Icon 1979
Ada 1980
C++ 1983
Objective-C 1983
Perl 1987
Haskell 1990
Python 1990
Ruby 2/24/93
Java 1995
JavaScript 1995
C# 2000
Scala 2003
F# 2005
Clojure 2007
Go 2008
Dart 2011
Rust 2012
Corelet 2013
Hack 2014
Swift 2014

## How are languages related to each other?

Some of the many attempts at a family tree of languages:

http://www.digibarn.com/collections/posters/tongues/

http://www.levenez.com/lang/

http://rigaux.org/language-study/diagram.html

## How many languages are there?

http://en.wikipedia.org/wiki/Alphabetical_list_of_programming_languages
(650+/-)

The Language List
http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm
"about 2,500", but lots of new ones missing

HOPL, the History of Programming Languages
http://hopl.murdoch.edu.au/ *(seems dead...)*
http://web.archive.org/web/20111205165034/http://hopl.murdoch.edu.au/ *(Internet Archive Wayback Machine)*
Over 8,000 but has things like "JAVA BEANS" and variants like both ANSI Pascal and ISO Pascal.

Bottom line: Nobody knows how many programming languages have been created!

## What languages are popular right now?

Measured by GitHub repositories:

adambard.com/blog/top-github-languages-2014/

Measured by job postings:

indeed.com/jobtrends

The TIOBE index (multiple factors):

www.tiobe.com/index.php/content/paperinfo/tpci/index.html

What *is* a good way to measure language popularity?

## How do languages help us?

Free the programmer from details

```
    int i = 5;
    x = y + z * q;
```

Detect careless errors

```
    int f(String s, char c);
    ...
    int i = f('i', "Testing");
```

Provide constructs to succinctly express a computation

```
    for (int i = 1; i <= 10; i++)
        ...
```

## How languages help, continued

- Provide portability

  Examples:

  - C provides moderate source-level portability.
  - Java was designed with binary portability in mind.

- Facilitate using a paradigm, such as functional, object-oriented, or logic programming.

## How are languages specified?

The specification of a language has two key facets:

- Syntax:
  Specifies the sequences of symbols that are valid programs in the language.

- Semantics:
  Specify the meaning of a sequence of symbols.

Some languages have specifications that are approved as international standards. Others are defined by nothing more than the behavior of a lone implementation.

## Syntax vs. semantics

Consider this expression:

```
a[i] = x
```

What are some languages in which it is syntactically valid?

In each of those languages, what is the meaning of it?

What are various meanings for these expressions?

```
x || y
x y
*x
```

## Building blocks

What are the building blocks of a language?

- Data types
- Operators
- Control structures
- Support for encapsulation
  - Functions
  - Abstract types / Classes
  - Packages / Modules
- Error / Exception handling
- Standard library

## What are qualities a language might have?

- Simplicity (“mental footprint”)
- Expressive power
- Readability of programs
- Orthogonality
- Reliability of programs
- Run-time efficiency
- Practical development project size
- Support for a style of programming

What are some tensions between these qualities?

## What factors affect popularity?

- Available implementations
- Documentation
- Community
- Vectors of “infection”
- Ability to occupy a niche
- Availability of supporting tools, like debuggers and IDEs
- Cost

## The philosophy of a language

What is the philosophy of a language? How is it manifested?

C
- Close to the machine
- Few constraints on the programmer
- High run-time efficiency
- "What you write is what you get."

C++
- Close to both machine and problem being solved
- Support object-oriented programming
- "As close to C as possible, but no closer." — Stroustrup

PostScript
- Page description
- Intended for generation by machines, not humans

What is the philosophy of Java?

## A Little UA CS History

## UA's language heritage

The UA CS department was founded by Ralph Griswold in 1971. (Hint: know this!)

Griswold was Head of Programming Research at Bell Labs before coming to UA.

Griswold and his team at Bell Labs created the SNOBOL family, culminating with SNOBOL4.

Griswold's interest and prominence in programming languages naturally influenced the course of research at UA.

## UA's heritage, continued

In the 1970s and 1980s UA Computer Science was recognized worldwide for its research in programming languages.

These are some of the languages created here:

| | |
|---|---|
| Cg | Seque |
| EZ | SIL2 |
| Icon | SL5 |
| Leo | SR |
| MPD | SuccessoR |
| Ratsno | Y |
| Rebus | Goaldi (in progress!) |

Along with language design, lots of work was focused on language implementation techniques.