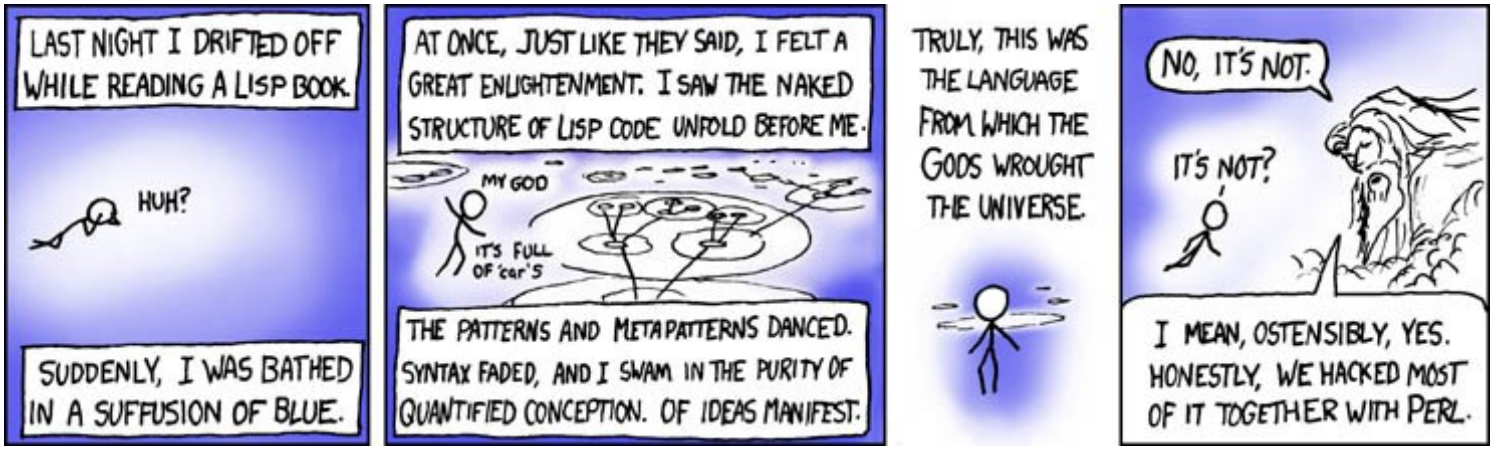
CS 430 Spring 2019

Mike Lam, Professor

Programming Languages



Opening challenge: define "programming language"

Overview

- *Programming language (PL)*
  - Tool for **formal** expression of problems and solutions
  - Audience: humans and machines
- General topics
  - **Syntax** (what a program looks like)
  - **Semantics** (what a program means)
  - **Implementation** (how a program executes)

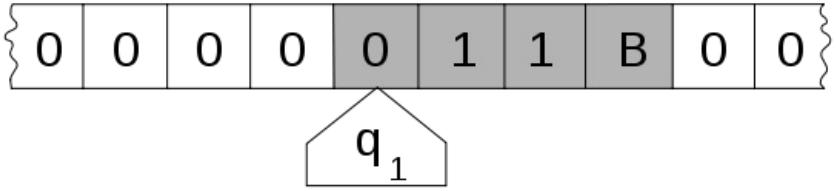
Why are PLs needed?

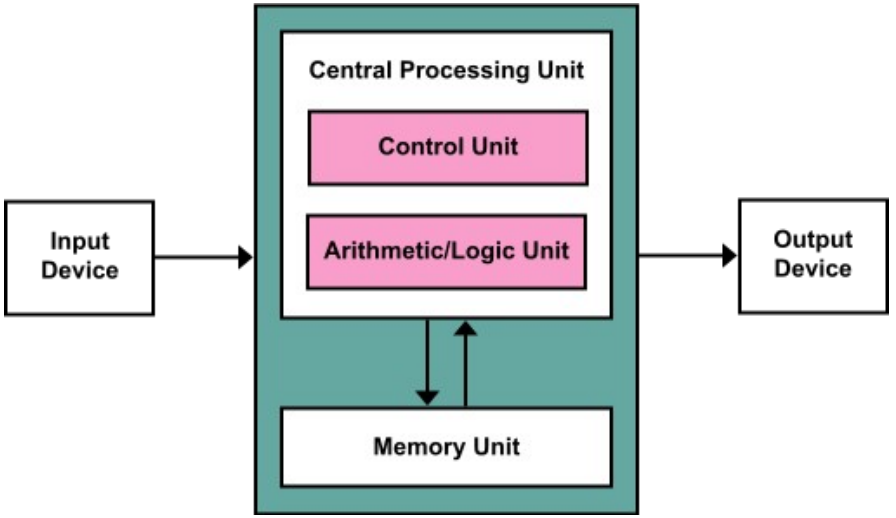
- Humans excel at approximate and contextual understanding
  - Imprecise language is often easier and quicker
  - Ex: "Meet you at El Charro at 6?"
  - vs. "I request your presence at 1580 S. Main St., Harrisonburg, VA, at 18:00 GMT-5 on 2019-01-10"

- Machines are not humans (and thus less forgiving)
  - They are (currently) slower and less accurate at language recognition and interpretation
  - Thus, programming in a natural language is a Bad Idea™

Why so many PLs?

- Surprising result: all languages are (theoretically) equivalent
  - A language is "Turing-complete" if it can compute anything computable by a Turing machine
  - Most modern languages are Turing-complete
- Also, most are designed for a von Neumann architecture
  - Data and program in the same memory
  - Fetch-decode-execute cycle





Why are there so many?

- Evolution over time
  - Just like human languages
- Deliberate design efforts
  - To address shortcomings of existing languages

Which language is best?

- It depends!

Our Goals

- Compare programming languages with regard to syntax and semantics
- Discuss language implementation issues and the tradeoffs involved
- Gain experience in learning new languages
- Gain experience using different language paradigms
  - E.g., scripting, functional, and logic-based

Course Design

- Mastery model
  - Course content divided into ~20 modules
  - Ungraded activities to **achieve** mastery
  - Graded assessments to **prove** mastery

- Schedule
  - 1-2 modules per week
  - Lectures and labs Monday and Wednesday (and sometimes Friday)
  - Assessment(s) due on Friday
  - Final grade is mean of all individual module grades

Tentative Schedule

Week	Date	Module(s)	CPL
1	Jan 7	01: Intro and Ruby 1 (R/P)	1
2	Jan 14	02: Ruby 2 (P)	
3	Jan 21	03: Haskell 1 (P)	15
4	Jan 28	04: Haskell 2 (P)	
5	Feb 4	05: Prolog 1 (P)	16
6	Feb 11	06: Prolog 2 (P)	
7	Feb 18	07: Syntax (B)	3
		08: Parsing (B)	4
8	Feb 25	09: Names and Bindings (R)	5
		10: Scope and Lifetime (B)	
	Mar 4	Spring Break	
9	Mar 11	11: Data Types (R)	6
		12: Type Checking (B)	
10	Mar 18	13: Expressions (B)	7
		14: Control Structures (R)	8
11	Mar 25	15: Parameters (B)	9
		16: Subprogram Invocation (R)	
12	Apr 1	17: Activations and Environments (B)	10
13	Apr 8	18: Abstraction and OOP (B)	11, 12
14	Apr 15	19: Concurrency and Error Handling (B)	13, 14
15	Apr 22	20: History (R)	2
		Review	

Module Types

- **Programming**
  - Learn a language by working on short projects
  - Assessed via automated testing (submit on Canvas)
  - No retakes!
- **Basic**
  - Learn via reading, lectures, videos, labs, discussions, problems
  - Assessed via closed-book tests, in-class on Friday
    - Option for re-take the following Friday
- **Reading**
  - Learn via reading
  - Assessed via open-book Canvas assessment due Friday
    - Option for re-take due the following Friday (except for M1!)

Learning Activities

- Module guides: lists of objectives
- Readings: Sebesta's "Concepts of Programming Languages" (CPL)
  - Reading is important
  - Some material will not be covered during class
- In-class lectures: focused on harder material
- Labs: in class, not graded
  - But strongly recommended!
- Web/Canvas resources

Tentative Schedule

Week	Date	Module(s)	CPL
1	Jan 7	01: Intro and Ruby 1 (R/P)	1
2	Jan 14	02: Ruby 2 (P)	
3	Jan 21	03: Haskell 1 (P)	15
4	Jan 28	04: Haskell 2 (P)	
5	Feb 4	05: Prolog 1 (P)	16
6	Feb 11	06: Prolog 2 (P)	
7	Feb 18	07: Syntax (B)	3
		08: Parsing (B)	4
8	Feb 25	09: Names and Bindings (R)	5
		10: Scope and Lifetime (B)	
	Mar 4	Spring Break	
9	Mar 11	11: Data Types (R)	6
		12: Type Checking (B)	
10	Mar 18	13: Expressions (B)	7
		14: Control Structures (R)	8
11	Mar 25	15: Parameters (B)	9
		16: Subprogram Invocation (R)	
12	Apr 1	17: Activations and Environments (B)	10
13	Apr 8	18: Abstraction and OOP (B)	11, 12
14	Apr 15	19: Concurrency and Error Handling (B)	13, 14
15	Apr 22	20: History (R)	2
		Review	

First 1/3: lots of **programming**, second 1/3: lots of **tests**, last 1/3: lots of **reading**

Questions?

Let's talk about PL

- Why should we want to study languages?

This material is also covered in Chapter 1 of your textbook.

Why PL?

- Increased capacity to express ideas
  - E.g., objects or associative maps in languages that don't explicitly provide them
- Improved background for choosing appropriate languages
  - We tend to choose things that are familiar, so it is advantageous to be familiar with many languages
- Increased ability to learn new languages
  - Practice helps, as does learning PL fundamentals
  - Also improves mastery of already-known languages

Why PL?

- Better understanding of implementation
  - Move beyond superficial differences between language syntax (whitespace, brackets, etc.)
  - Helps with program debugging
- Overall advancement of computing
  - Broader knowledge enables informed trends
  - CS does not benefit from "language ghettos" or flamewars
  - Hindsight: what if ALGOL 60 had become more popular than Fortran in the 1960s?

Why PL? (the real reasons)

- Knowing more languages looks good on your resume
- Knowing PL theory makes you a more valuable employee
- You get to brag about all the stuff you know
  - We're using two "hip"/"cult" languages this semester
- It's fun!
  - (I think so, anyway...)

How do we evaluate languages?

- Readability
  - How easy is it to understand already-written code?
- Writability
  - How easy is it to write clear, efficient code?
- Reliability
  - How easy is it to write programs that adhere to specifications?

This is a Sudoku solver in Perl:

```
$_=$`.$_.$'.<>;split//;${/[@_[map{$i-($i="@-")%9+$_,9*$_+$i%
9,9*$_%26+$i-$i%27+$i%9-$i%3}0..8]]/o||do$0}for/0/||print..9
```

(or is it?)

Evaluating Languages

- Simplicity (few basic constructs, minimal overloading)
- Orthogonality (independence of features, feature symmetry)
- Data types (expressive without being redundant)
- Syntax design (consistency, sensible keywords)
- Support for abstraction (subprograms, data structures)
- Expressivity (convenience, "elegance")
- Type checking (strict is safer, but cost vs. benefit is debatable)
- Exception handling (early detection, clean handling)
- Restricted aliasing (make it apparent)
- Standardization (respected organization, appropriate time)

Evaluating Languages

Table 1.1 Language evaluation criteria and the characteristics that affect them

	CRITERIA		
Characteristic	READABILITY	WRITABILITY	RELIABILITY
Simplicity	•	•	•
Orthogonality	•	•	•
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

Evaluating Languages

- Various costs
  - Programmer training
  - Code writing and debugging
  - Compile time
  - Execution time
  - Runtime system
  - Maintenance
  - Porting
- Tradeoffs exist between these criteria and costs
  - Language designs represent points on these spectrums

Language Categories

- Traditional bins:
  - Procedural/imperative (assembly, Fortran, COBOL, ALGOL, C)
  - Functional (Lisp, Scheme, Haskell)
  - Logic- or rule-based (Prolog, Make)
  - Object-oriented (Smalltalk, C++, Java, Ruby)
- Other bins:
  - Visual (Visual Basic, Adobe Flash)
  - Scripting (Perl, Javascript, Python, Ruby)
  - Markup or metadata (HTML, LaTeX)
  - Educational (Scratch)
  - Special-purpose or domain-specific

Contexts

- Context matters!
  - Languages do not exist in a vacuum

Context: Programming Domains

- Scientific
  - Primary concern: efficiency (speed)
- Business
  - Primary concern: data processing and formatting
- Artificial intelligence
  - Primary concern: symbolic computation
- Systems
  - Primary concern: efficiency, low-level access, and portability
  - Value of language safety is hotly debated
- Web
  - Primary concern: presentation and ease of development

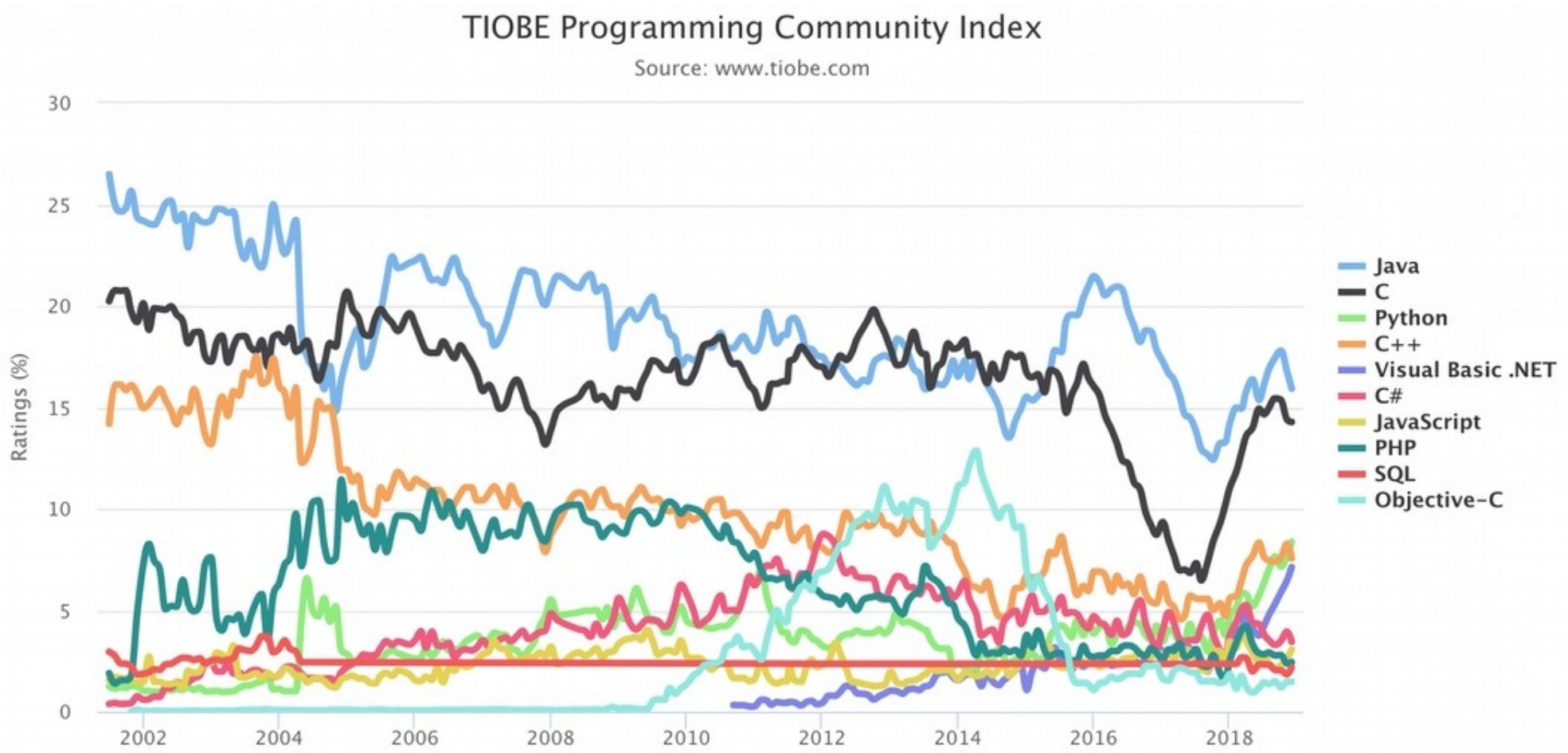
Context: PL Design Influences

- Hardware/architecture design shifts
  - Historic prevalence of imperative/procedural languages that closely match the hardware (von Neumann architecture)
  - Cheaper hardware → higher-level languages
- Software development methodology shifts
  - Shift from procedure-oriented to data-oriented
  - Better software engineering practices → desire for “safer” languages
  - Agile programming and rapid prototyping languages
- Social, cultural, and political shifts
  - Millennial and post-millennial generation cultures (web languages and frameworks)

Relative Popularity

- What do you suppose was the fastest-growing language in 2018?
  - (according to the TIOBE index, anyway...)

Relative Popularity



Fastest growing language of 2018? VB.NET!

(Python and C were close seconds)

Historical Popularity

Programming Language	2018	2013	2008	2003	1998	1993	1988
Java	1	2	1	1	16	-	-
C	2	1	2	2	1	1	1
C++	3	4	3	3	2	2	5
C#	4	5	7	11	-	-	-
Python	5	7	6	12	27	16	-
Visual Basic .NET	6	13	-	-	-	-	-
JavaScript	7	9	8	7	20	-	-
PHP	8	6	4	5	-	-	-
Perl	9	8	5	4	3	9	-
Ruby	10	10	9	19	-	-	-
Objective-C	18	3	44	46	-	-	-
Ada	27	16	17	14	6	7	2
Lisp	31	12	14	13	8	5	3
Pascal	128	14	19	97	10	3	6

First New Language: Ruby

- Ruby is a **dynamically-typed**, **pure object-oriented**, **interpreted** scripting language

```
puts “Hello world!”    # this is a complete program!
```

There is a lab posted on the website to help you learn Ruby.

The first PA is also posted.

On Wednesday and Friday we will have open lab time to work on learning Ruby.

Learning New Languages

- Write code!
  - Learning *about* a language ≠ learning the language
- Ideas:
  - Do the provided labs!
  - Do the programming assignments
  - Re-write your CS 149 projects in the new language
  - Re-write a hobby project in the new language
  - Solve problems on a site like Kattis, HackerRank, etc.

Good luck!

- For Wednesday:
  - Start learning Ruby (lab posted)
  - Take the intro survey
  - Take the M1 reading test if you wish (posted soon)
- Have a great semester!