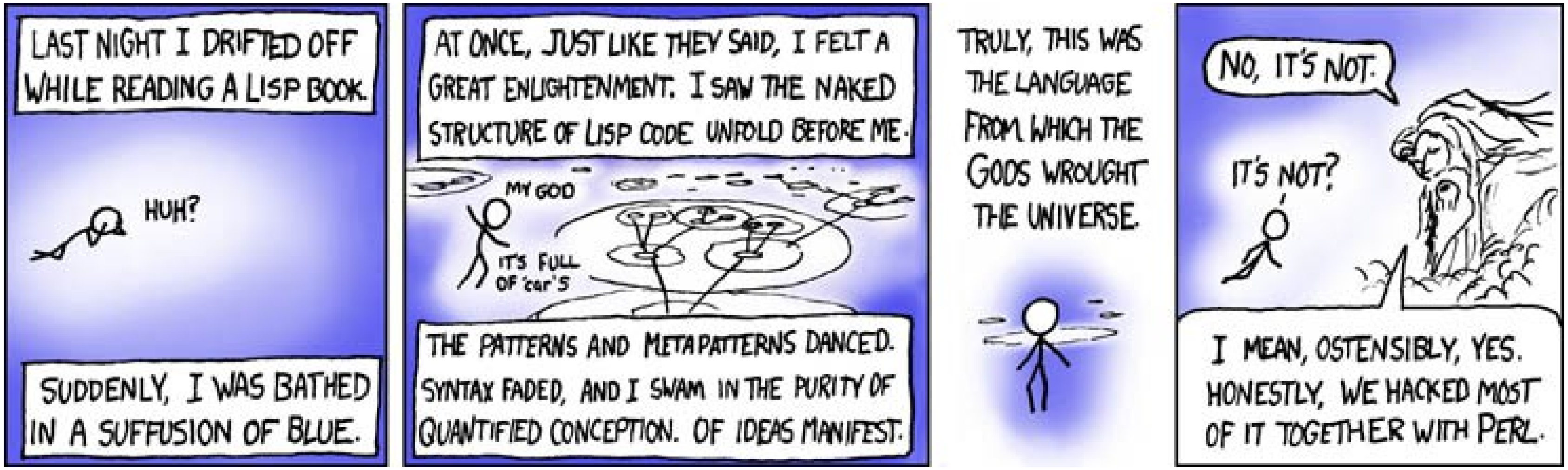
CS 430
Spring 2022

Mike Lam, Professor

Programming Languages


LAST NIGHT I DRIFTED OFF WHILE READING A LISP BOOK.
HUH?
SUDDENLY, I WAS BATHED IN A SUFFUSION OF BLUE.
AT ONCE, JUST LIKE THEY SAID, I FELT A GREAT ENLIGHTENMENT. I SAW THE NAKED STRUCTURE OF LISP CODE UNFOLD BEFORE ME.
MY GOD
IT'S FULL OF 'car'S
THE PATTERNS AND METAPATTERNS DANCED. SYNTAX FADED, AND I SWAM IN THE PURITY OF QUANTIFIED CONCEPTION. OF IDEAS MANIFEST.
TRULY, THIS WAS THE LANGUAGE FROM WHICH THE GODS WROUGHT THE UNIVERSE.
NO, IT'S NOT.
IT'S NOT?
I MEAN, OSTENSIBLY, YES. HONESTLY, WE HACKED MOST OF IT TOGETHER WITH PERL.

Opening challenge: how many programming languages can you name?

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)

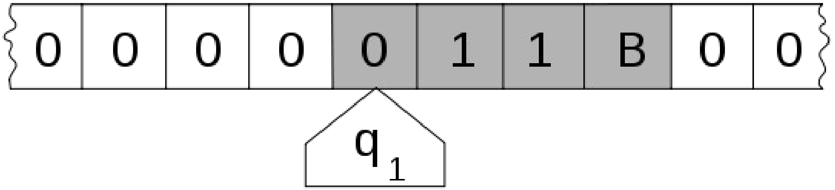
Why are PLs needed?

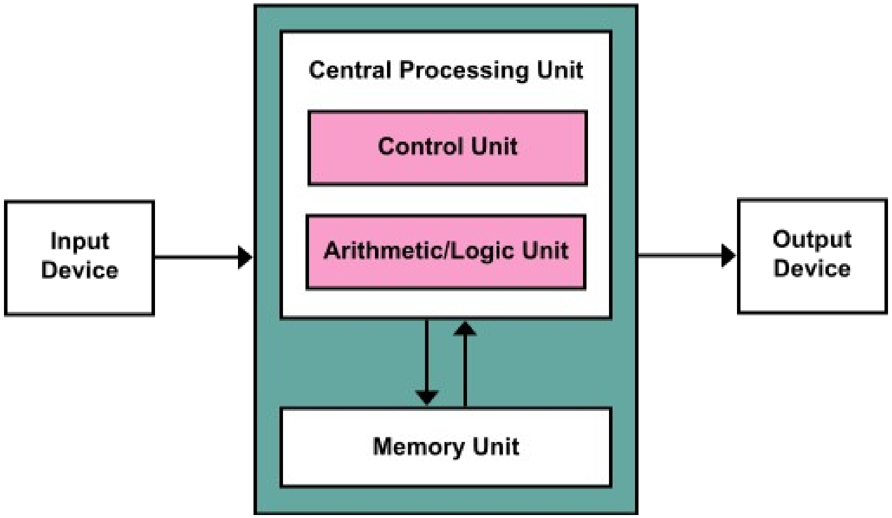
- Humans excel at approximate and contextual understanding
  - Imprecise language is often easier and quicker
  - Ex: "Meet you at El Charro at 6?"
  - vs. "I request your presence at 1480 S. Main St., Harrisonburg, VA, at 18:00 GMT-5 on 2022-01-19"

- Machines are not humans
  - Less capable of correctly dealing with imprecision
  - Thus, programming in a natural language is a Bad Idea$^{\text{TM}}$

Surprisingly homogeneous

- Almost 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
- Humans are creative
  - And opinionated!

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 declarative

Course Design

- Mastery model
  - Course content divided into ~20 modules
  - “Lightly graded” activities to **achieve** mastery
  - Graded assessments to **prove** mastery

- Schedule
  - 1-2 modules per week
  - Lectures and labs Tuesday and Thursday
  - Assessment(s) due on Friday
  - Final grade is mean of all individual module grades
  - Final exam: a flash talk on a language not covered in this course

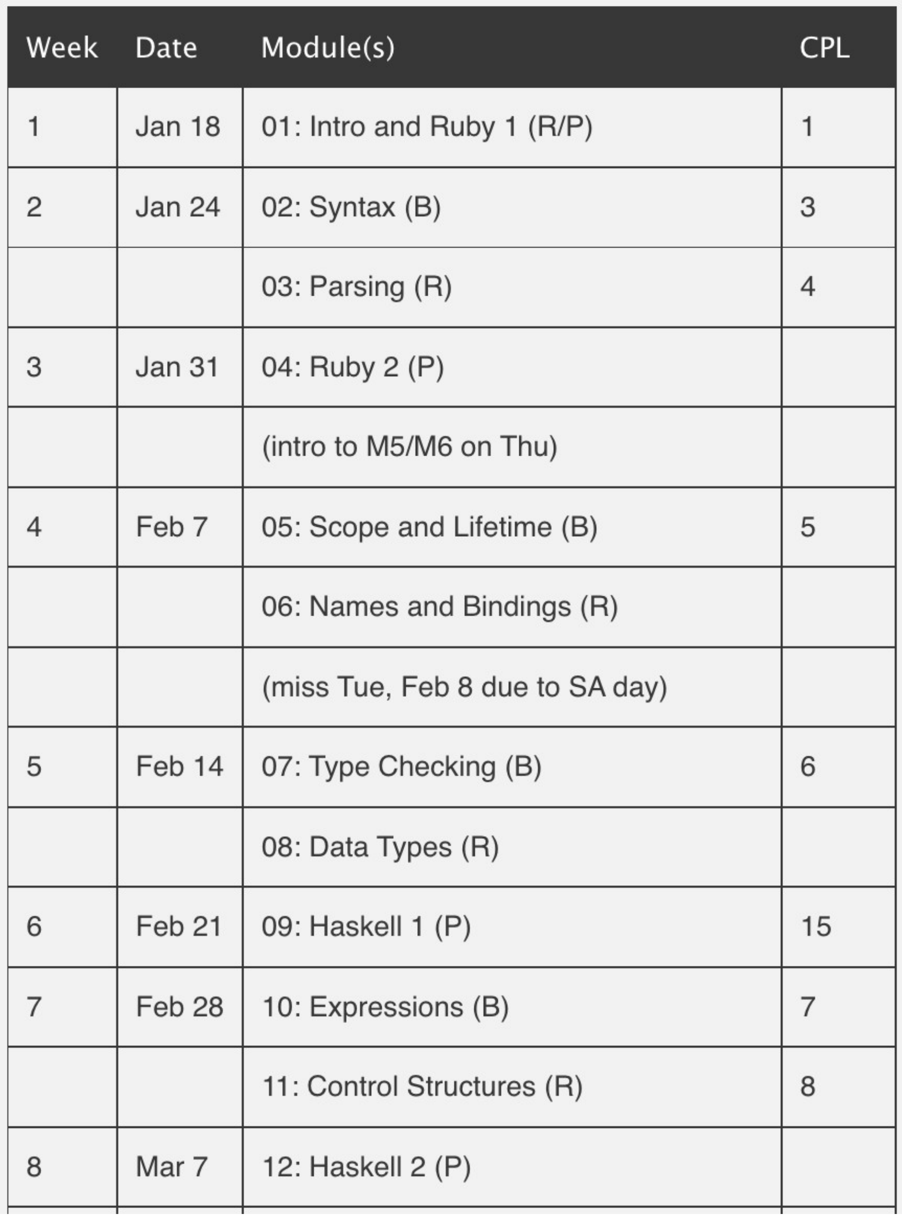
Module Types

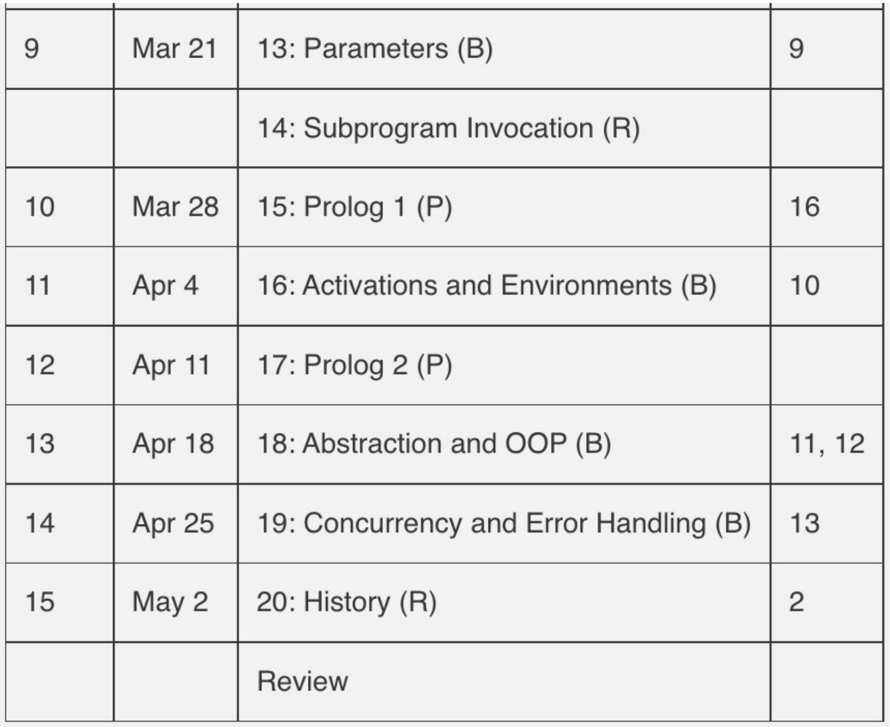
- **Basic (B)**
  - Learn via readings, lectures, and labs (Mon-Thu)
  - Assessed via Canvas quiz on Friday
    - Option for second try the following Friday
- **Reading (R)**
  - Learn via reading
  - Assessed via Canvas quiz due Friday
    - Option for second try due the following Friday
- **Programming (P)**
  - Learn a language by working on labs and short projects
  - Assessed via automated testing (submit on Canvas, due Friday)
  - No retakes!

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
- In-class labs: submitted on Canvas, graded “lightly” (except M1)
  - Reference solutions will often be posted (but not always)
- Web/Canvas resources
- Watch for upcoming assessments and plan around them
  - Think about due dates in other courses as well
  - R and P module deliverables often can be completed early

Tentative Schedule





Week	Date	Module(s)	CPL
1	Jan 18	01: Intro and Ruby 1 (R/P)	1
2	Jan 24	02: Syntax (B)	3
		03: Parsing (R)	4
3	Jan 31	04: Ruby 2 (P)	
		(intro to M5/M6 on Thu)	
4	Feb 7	05: Scope and Lifetime (B)	5
		06: Names and Bindings (R)	
		(miss Tue, Feb 8 due to SA day)	
5	Feb 14	07: Type Checking (B)	6
		08: Data Types (R)	
6	Feb 21	09: Haskell 1 (P)	15
7	Feb 28	10: Expressions (B)	7
		11: Control Structures (R)	8
8	Mar 7	12: Haskell 2 (P)	
9	Mar 21	13: Parameters (B)	9
		14: Subprogram Invocation (R)	
10	Mar 28	15: Prolog 1 (P)	16
11	Apr 4	16: Activations and Environments (B)	10
12	Apr 11	17: Prolog 2 (P)	
13	Apr 18	18: Abstraction and OOP (B)	11, 12
14	Apr 25	19: Concurrency and Error Handling (B)	13
15	May 2	20: History (R)	2
		Review	

Class Policies

- Masks must be worn in class
  - Must cover both your mouth and nose
  - N95 or KN95 recommended over cloth and surgical
  - *(There are spare masks in the classroom if you need one)*
- No food or drink is allowed
  - *(Quick sips are ok w/ me – stay hydrated!)*
- If you are ill, **please stay home**
  - Contact me ASAP regarding missed class
- These policies may change
  - Changes will be announced via Canvas message

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
  - 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
- 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

- 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
  - Safety and security are a rapidly-growing concerns
- Web
  - Primary concern: presentation and ease of development

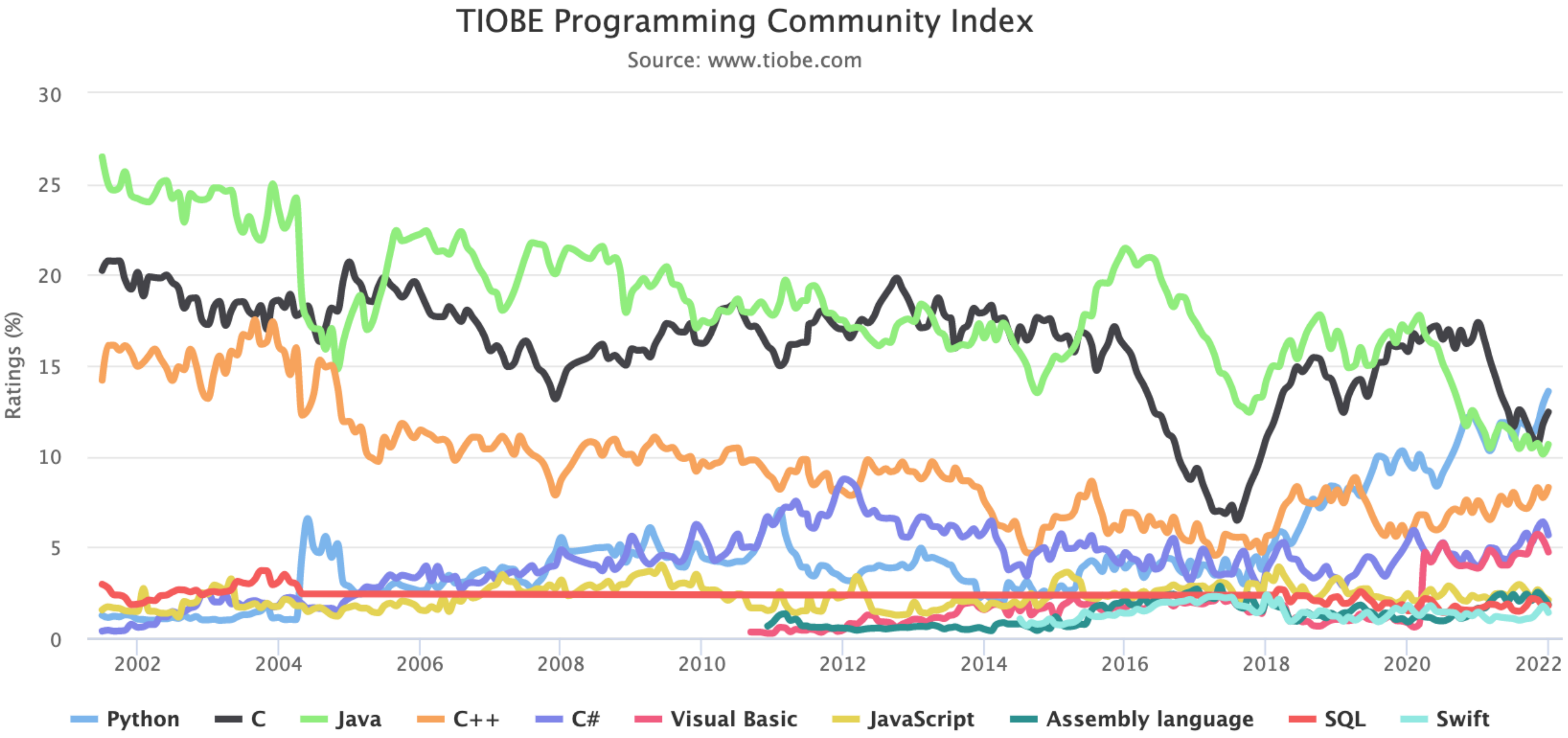
Context: PL Design Influences

- 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 2021?
  - (according to the TIOBE index, anyway...)

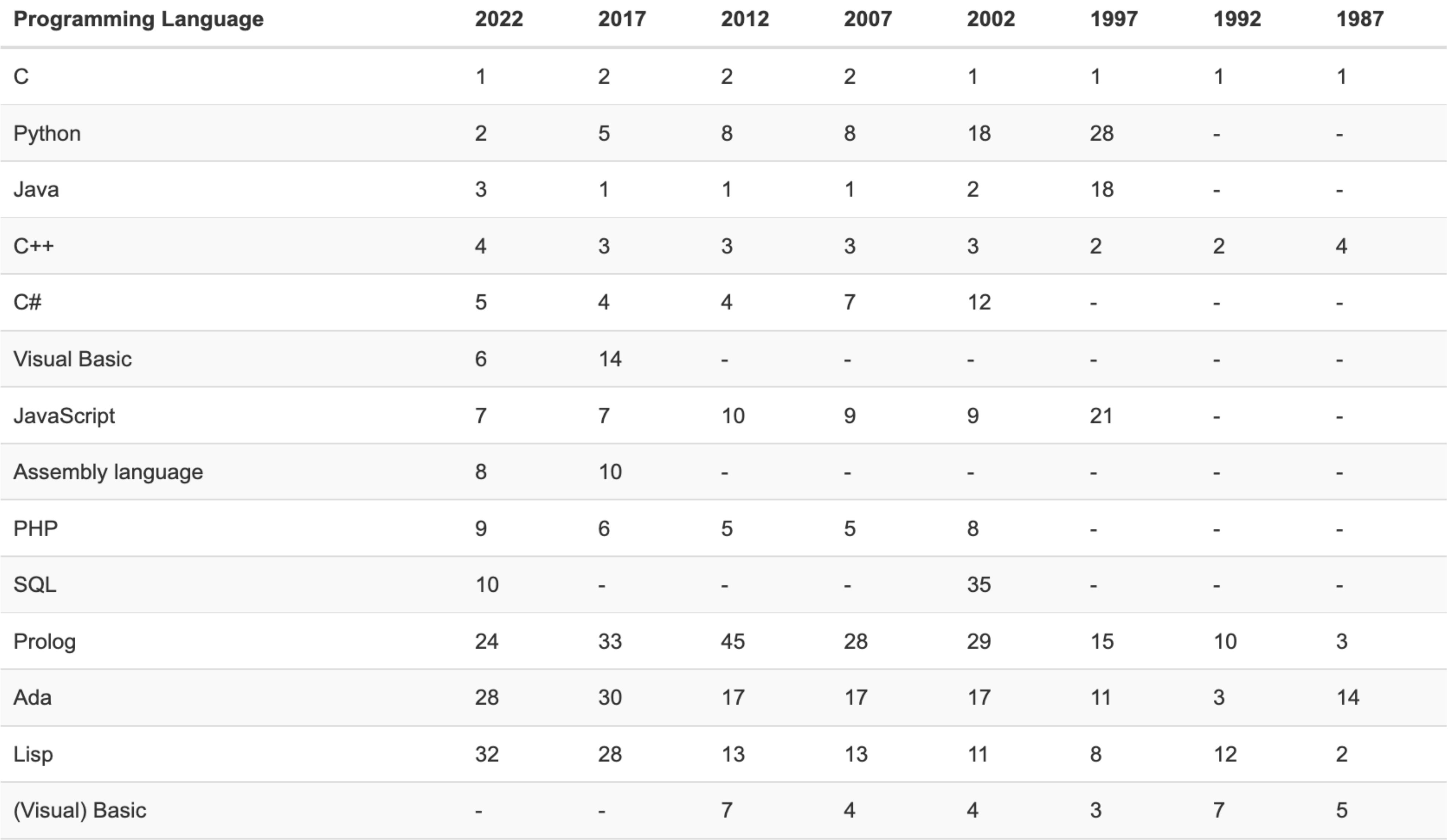
Relative Popularity



Fastest growing language of 2021? Python!

(for two years in a row now)

Historical Popularity



Programming Language	2022	2017	2012	2007	2002	1997	1992	1987
C	1	2	2	2	1	1	1	1
Python	2	5	8	8	18	28	-	-
Java	3	1	1	1	2	18	-	-
C++	4	3	3	3	3	2	2	4
C#	5	4	4	7	12	-	-	-
Visual Basic	6	14	-	-	-	-	-	-
JavaScript	7	7	10	9	9	21	-	-
Assembly language	8	10	-	-	-	-	-	-
PHP	9	6	5	5	8	-	-	-
SQL	10	-	-	-	35	-	-	-
Prolog	24	33	45	28	29	15	10	3
Ada	28	30	17	17	17	11	3	14
Lisp	32	28	13	13	11	8	12	2
(Visual) Basic	-	-	7	4	4	3	7	5

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.

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 project is also posted.

Once you finish the lab you will have the tools necessary to complete the project quickly.

(skim through the Array class documentation for other helpful methods!)

Ruby is very expressive

- All of the following snippets are equivalent!

```
i = 1
while i <= 9
  print i
  i += 1
end
```

```
i = 1
until i > 9
  print i
  i += 1
end
```

```
i = 1
loop do
  print i
  i += 1
  break if i > 9
end
```

```
for i in 1..9 do
  print i
end
```

```
1.upto(9).each do |i|
  print i
end
```

```
(1..9).each do |i|
  print i
end
```

“functional style”

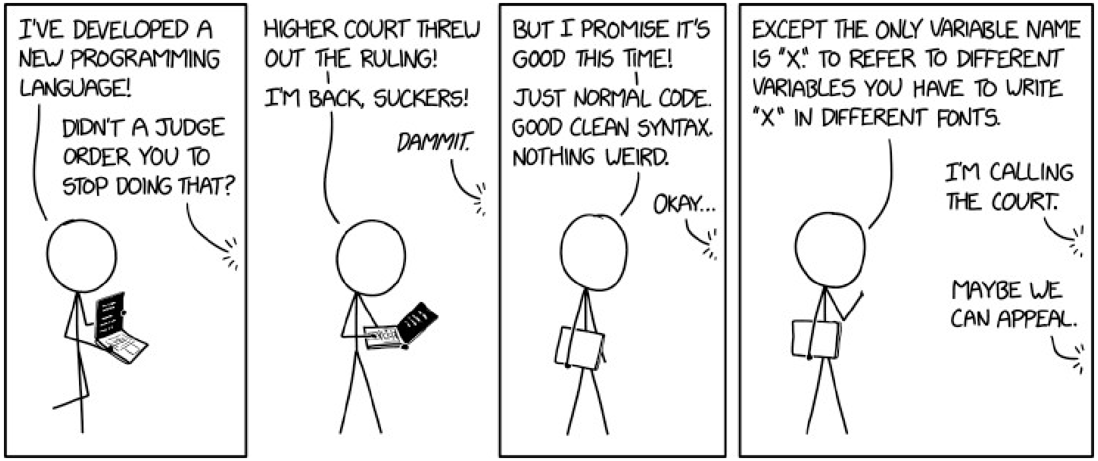
```
(1..9).each { |i| print i }
```

```
print (1..9).to_a.join
```

Optional challenge: write all P1 functions using a single line of code!

Module 1

- Course survey (1 pt)
- Reading quiz “M1 Quiz A” (4 pts)
  - Re-take available through next week
- Project 1 “M1: Ruby 1” (5 pts)
  - Use the remaining time today to work on this


I'VE DEVELOPED A NEW PROGRAMMING LANGUAGE!
DIDN'T A JUDGE ORDER YOU TO STOP DOING THAT?
HIGHER COURT THREW OUT THE RULING!
I'M BACK, SUCKERS!
DAMMIT.
BUT I PROMISE IT'S GOOD THIS TIME!
JUST NORMAL CODE. GOOD CLEAN SYNTAX. NOTHING WEIRD.
OKAY...
EXCEPT THE ONLY VARIABLE NAME IS "X". TO REFER TO DIFFERENT VARIABLES YOU HAVE TO WRITE "X" IN DIFFERENT FONTS.
I'M CALLING THE COURT.
MAYBE WE CAN APPEAL.

https://xkcd.com/2309/