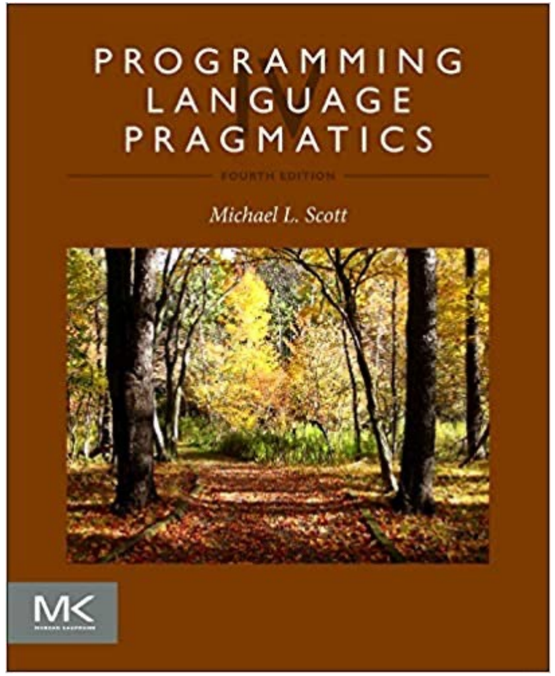
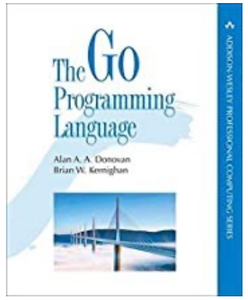
Programming Languages

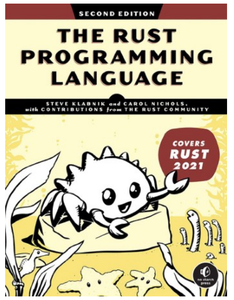
CS245

Things to Know

- Textbook
  - Programming Language Pragmatics, v4
    - by M Scott
      - 17 chapters, 9 covered and not all of those
- Also
  - The Go Programming Language
    - Donovan & Kernighan
  - The Rust Programing Language
    - Klabnik & Nichols



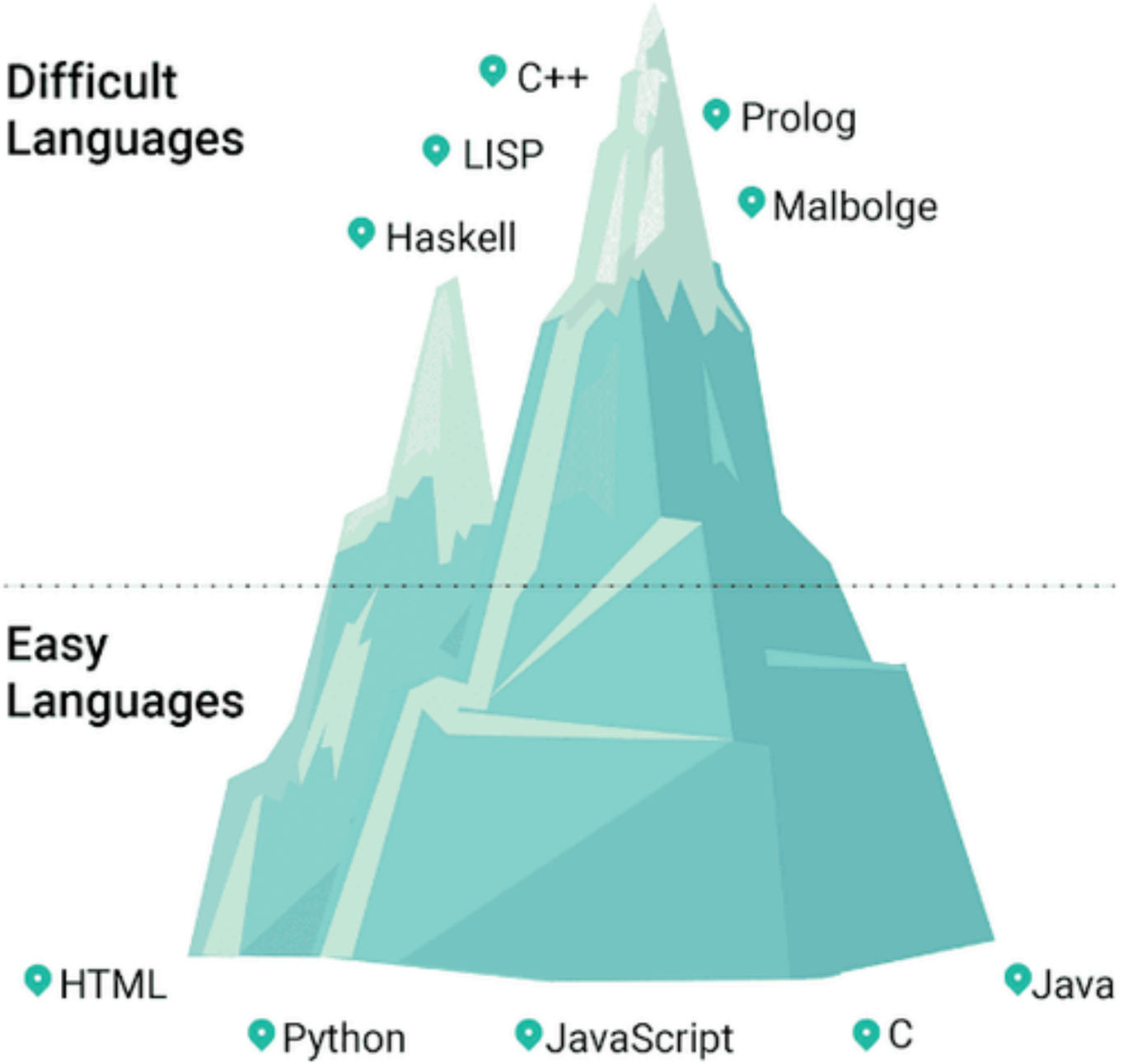




Programming Languages

- Why?
  - because it is required for the major
- Why is it required?
  - if a PL is “Turing Complete” it does not matter what language you use. All modern PLs are.

  - Grace Hopper — if you program in a better language you can be more efficient



https://www.interviewkickstart.com/blog/hardest-and-easiest-programming-languages-to-learn

Class WebSite

https://cs.brynmawr.edu/cs245

- Will have all homework, important dates, etc

- Lecture notes — I will post PDF “notes”. Literally my notes to myself.
  - This will likely be the only powerpoint for the class

- Tests — 2 midterms and a final. All will be open book, open notes, open computer, closed mouth.
  - midterms will be “take where you want, but on a given date”
  - final — similar idea.

- Homeworks — approximately 6 through the semester

- Lab — The first few will be formally in lab room. These will be graded on a “did you hand in something that is at least semi-correct”.

- Class participation -- will be assessed on a "readiness to participate" rather than actual participation. Readiness will be assessed with occasional 5 minute quizzes at start of class.

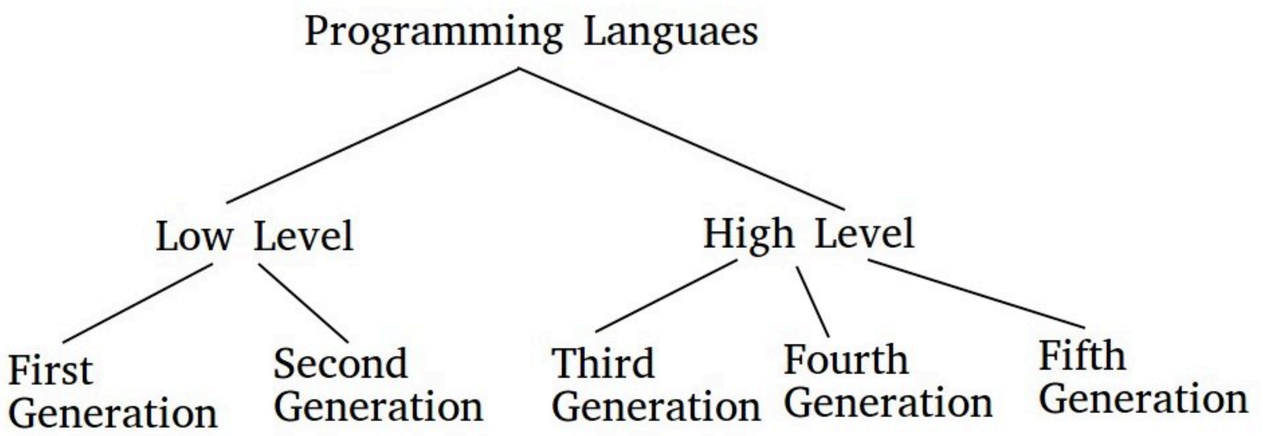
Lab

- Both this week and next week.

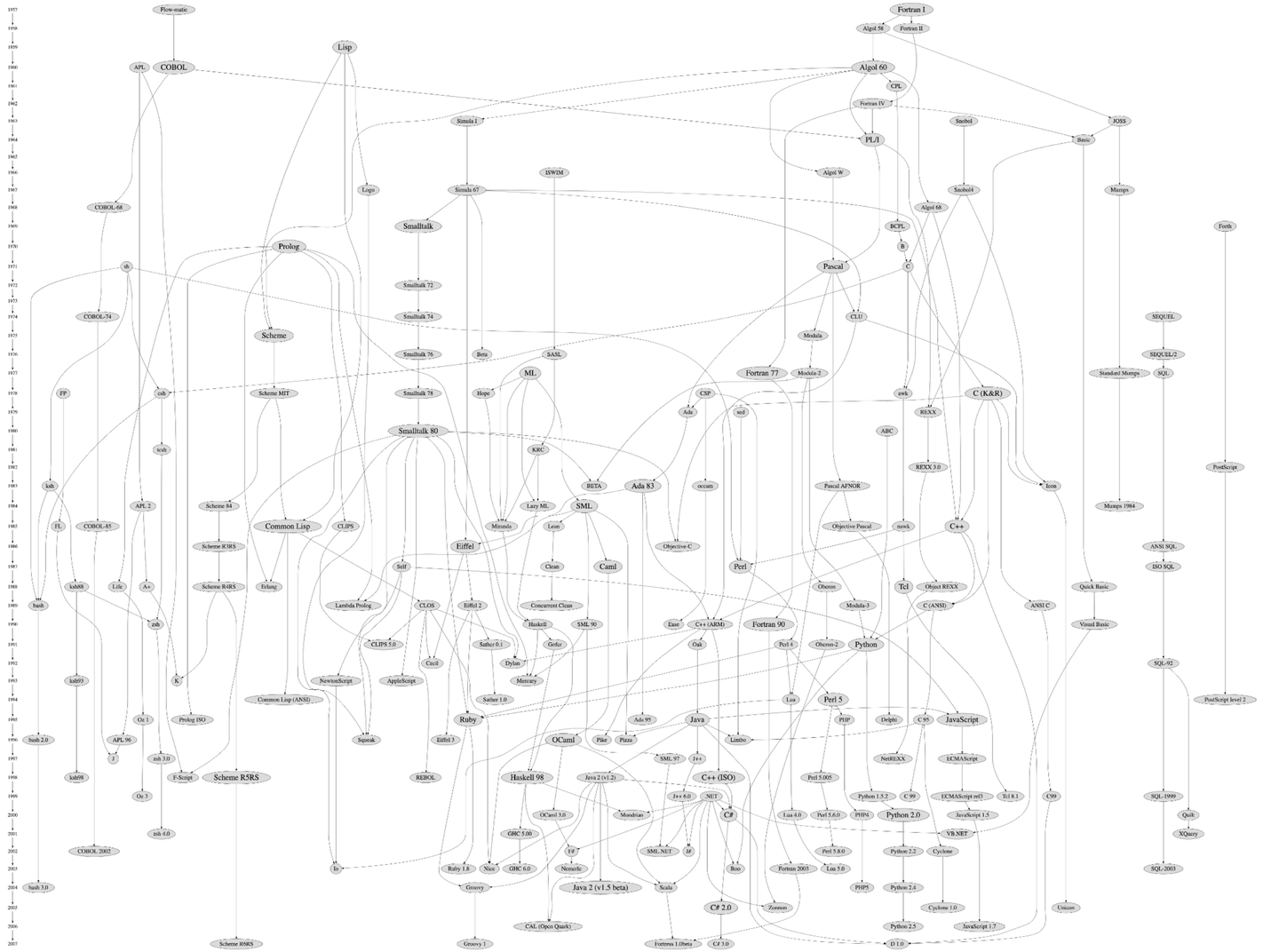
- If you have timing issues, not a problem to arrive late.
  - Formally, labs are not due until midnight of the lab date.
  - Think of lab as
    - an office hour in which I am sitting in 231
    - I have given you a task and you should work on that task for 80 minutes

Goals

- Learn questions to ask, and how to evaluate answers, for choosing language appropriate to problem
- Improve ability to learn new programing languages
  - In my career: Basic, PL/1, Pascal, (rascal, spss, sas), C, Lisp, Prolog, Visual-C, Perl, Python, Visual-Basic, Java, SQL, Objective-C, PHP, Javascript, Kotlin, Go, Elixir, Swift, Rust (and probably a dozen others)
- Increase ways in which you can express and implement programs
- Understand why and wherefore of “obscure” language features



- First Generation
  - Machine language -- literally working with 0 and 1
- Second Generation
  - Assembly language -- write commands that are directly supported by CPU
- **Third Generation**
  - **Most PLs that you will ever work with: C, Java, ...**
- Fourth Generation
  - Giving instructions to VM that specify what, not how: SQL, R(?)
- Fifth Generation
  - AI stuff: Prolog, Lisp(?)



https://github.com/stereobooster/programming-languages-genealogical-tree

A Really Brief Genealogy of PLs

First Compiler/ Interpreter	Language	Lead Designer	Progeny
1952	A-0	Hopper	FLOW-MATIC
1955	FLOW-MATIC	Hopper	COBOL
1957	**Fortran**	Backus	ALGOL (Fortran influences, directly or indirectly, every other language on this page excepting Lisp, COBOL and APL)
1958	ALGOL	committee	BCPL, Pascal
1958	Lisp	McCarthy	Scheme, all functional languages, Ruby
1959	**COBOL**	Hopper + committee	
1964	APL	Iverson	(small family of descendants)
1964	**BASIC**	Kemeny / Kurtz	Apple and Microsoft Basics, scripting language in MS Office, Lotus Notes and many others
1964	PL/I	IBM	
1966	BCPL	Richard	B
1967	Perl	Wall	influences many, Ruby
1969	B	Thompson	C
1970	Pascal	Wirth	Modula-2
1972	**C**	Ritchie	C++, JavaScript (and most later languages), Go
1972	Smalltalk	Kay, Ingalls, Goldberg	C++, JavaScript (via Self), all object languages
1975	Scheme	Steele and Sussman	JavaScript, all functional languages
1978	Modula-2	Wirth	Modula-3
1983	**C++**	Stroustrup	Java (and numerous others)
1987	Self	Ungar, Smith	JavaScript
1989?	Modula-3	committee	Java, Python
1991	Python	van Rossum	Ruby
1995	**Java**	Gosling	(most later languages)
1995	JavaScript (neé Mocha)	Eich	
1995	PHP	Lerdorf	
1995	Ruby	Matsumoto	elixir

http://www.martinrinehart.com/pages/genealogy-programming-languages.html

Java

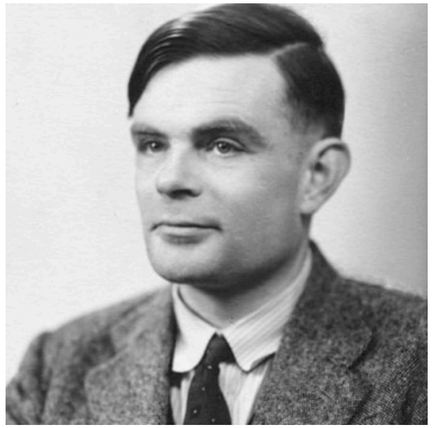
Why is Java the first language taught at BM

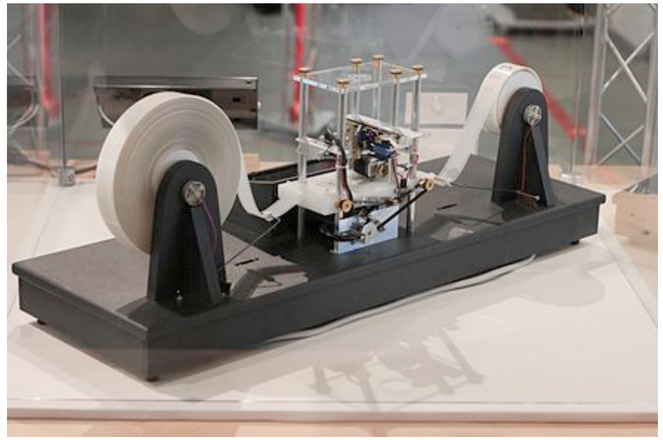
- Plusses
  - large, well-organized libraries
  - clean, consistent syntax
  - Easily available instructional support
  - Widely used

- Minuses
  - OO is big hurdle — have to “talk around it”
  - Comically wordy

Church Turing Thesis

- any function whose values can be computed by an algorithm can be computed by a Turing machine, and therefore that if any real-world computer can simulate a Turing machine, it is Turing equivalent to a Turing machine





Rust is a multi-paradigm, general-purpose programming language that emphasizes performance, type safety, and concurrency. It enforces memory safety—ensuring that all references point to valid memory—without requiring the use of a garbage collector or reference counting present in other memory-safe languages. To simultaneously enforce memory safety and prevent concurrent data races, its "borrow checker" tracks the object lifetime of all references in a program during compilation. Rust borrows ideas from functional programming, including static types, immutability, higher-order functions, and algebraic data types. It is popularized for systems programming

Alef, C#, C++, Cyclone, Elm[5], Erlang, Haskell, Limbo, Newsqueak, OCaml, Ruby, Scheme, Standard ML, Swift

https://en.wikipedia.org/wiki/Rust_(programming_language)

improve programming productivity in an era of multicore, networked machines and large codebases.[20] The designers wanted to address criticism of other languages in use at Google, but keep their useful characteristics:[21]

- Static typing and run-time efficiency (like C)
- Readability and usability (like Python or JavaScript)[22]
- High-performance networking and multiprocessing

Its designers were primarily motivated by their shared dislike of C++

C, Oberon-2, Limbo, Active Oberon, communicating sequential processes, Pascal, Oberon, Smalltalk, Newsqueak, Modula-2, Alef, APL, BCPL, Modula, occam

https://en.wikipedia.org/wiki/Go_(programming_language)

Why not .....

The complexity of C++ (even more complexity has been added in the new C++), and the resulting impact on productivity, is no longer justified. All the hoops that the C++ programmer had to jump through in order to use a C-compatible language make no sense anymore -- they're just a waste of time and effort. Go makes much more sense for the class of problems that C++ was originally intended to solve.

Bruce Eckel -- a founding member of the ANSI/ISO C++ standard committee

Hello World

```rust
// Rust

fn main() {
    println!("Hello, world!");
}
```

```
rustc hw.rs
hw

OR

cargo run // assuming cargo is being used
```

```go
//GO

package main
func main() {
    println("hello geoff!");
}
```

```
go run hw.go

OR

go build hw.go
hw
```

For next class

- If you could be a programming language, which one would you be
  - Why?
  - Why is that language so named?
    - Do not use: Java, C, Python, Fortran, Cobol, Javascript, Rust, Go.
- Read
  - Scott 1.1-1.4
- This Weekend -- Watch (at least the first 30 minutes)
  - "The worst programming language ever"
    - https://www.youtube.com/watch?v=vcFBwt1nu2U
    - On Tues Sep 12 -- one statement that you did not understand or thought was really funny

Functional and Imperative programming

- Imperative
  - programming by side effect
    - procedures that return nothing (in Java void)
  - lots of variables whose values are set and change frequently

- Functional
  - No variables
    - there are things that look like variables but they are better though of as constants
      - What is the difference between a variable and a constant whose value you can change?
  - Functions always return values, it is why they are executed
  - Functions are only dependent on their arguments
    - Programs can be provably correct (usually of academic interest only)