CS 430
Spring 2022

Mike Lam, Professor

Grace Hopper (1906-1992)

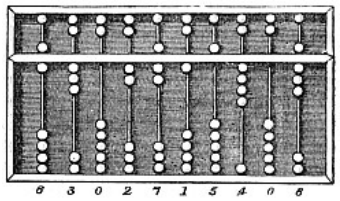
Dennis Ritchie (1941-2011)

History of Programming Languages

All media taken from Sebesta or Wikipedia unless stated otherwise

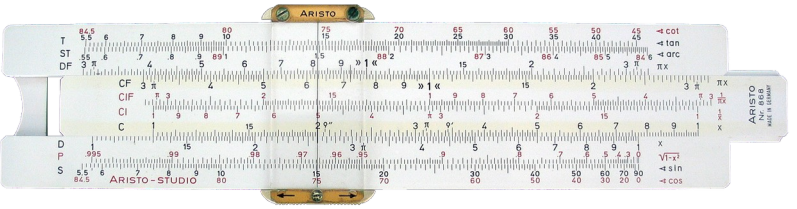
Historical Computing Devices

- Abacus - 算盘 (2nd century BCE)
  - Basic counting and arithmetic
- Antikythera mechanism (2nd century BCE)
  - Calculated astronomical motion
- Slide rules (17th century CE)
  - More complex arithmetic enabled by logarithms



Abacus

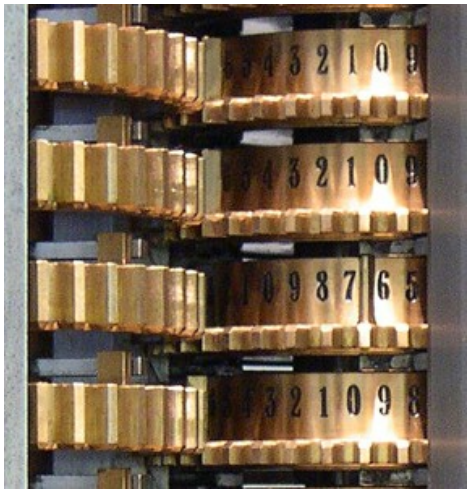
Antikythera mechanism



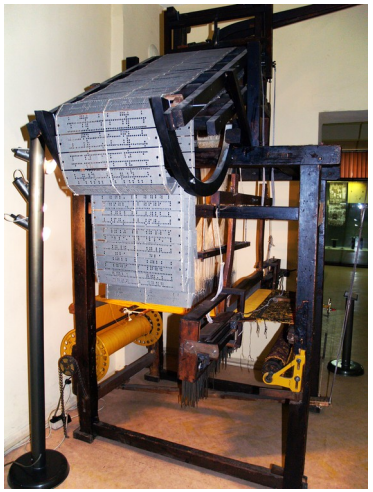
Slide rule

Mechanical Computers

- Difference engine (19th century)
  - Designed by Charles Babbage, an English mathematician
  - Ada Lovelace (one of his collaborators) is often credited as the first computer programmer
  - https://www.youtube.com/watch?v=BlbQsKpq3Ak
- Jacquard machines (19th century)
  - https://www.youtube.com/watch?v=K6NgMNvK52A
- Fire control computers (mid-20th century)
  - https://www.youtube.com/watch?v=s1i-dnAH9Y4



Difference engine

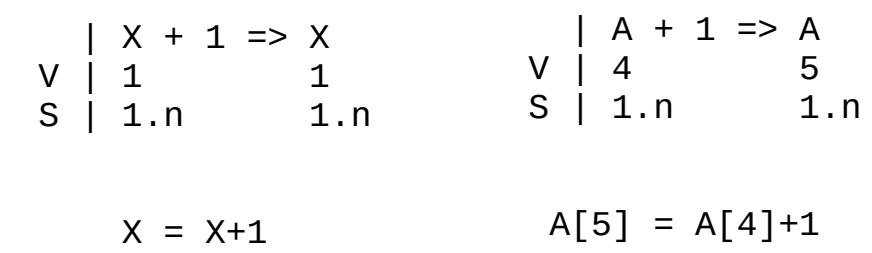


Jacquard machine

Fire control computer

Konrad Zuse's Plankalkül (1945)

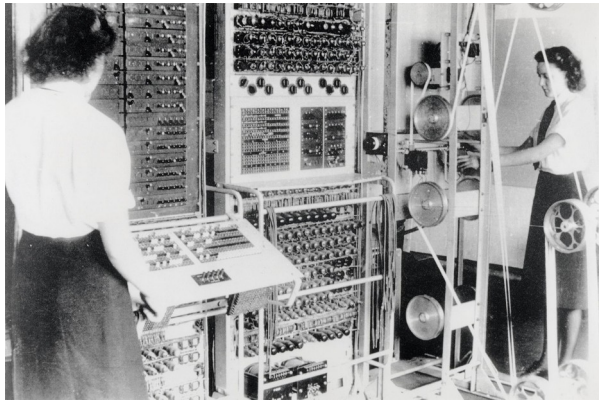
- "Program Calculus" or "Plan Calculus"
- Designed for Zuse's electromechanical Z4 machine
- Many innovative concepts
  - Data types and arrays
  - Iteration and control flow
- Verbose written style
  - Originally 2-dimensional
- Not widely known at the time
  - World War II obscured parallel development efforts in Germany, Great Britain, and the U.S.



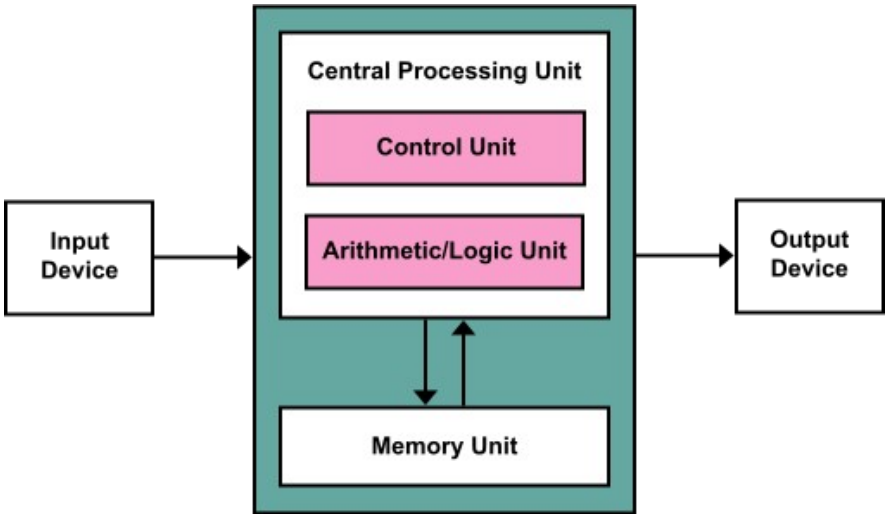
von Neumann Architecture

- Programs stored in memory
- Machine-specific **opcodes** and **operands** encoded in binary
- Fetch-Decode-Execute cycle
- Heavily influenced early programming languages

The earliest computers were "programmed" manually by entering opcodes and operands using switches or tape



Colossus machine at Bletchley Park in 1943



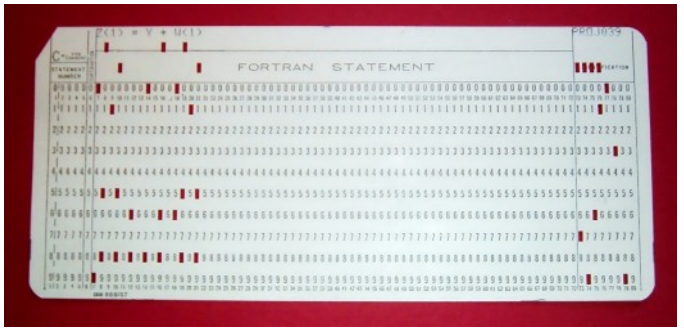
Early Digital Computing ('40s-'50s)

- Every machine had a different set of binary instructions and data addressing modes
  - Low readability and writability; programming was HARD!
  - Programs were very rigid because of explicit memory addresses
  - Very little portability between machines
  - No hardware support for floating-point arithmetic
- Interpreted “pseudocodes” helped address these issues
  - Sometimes called “automatic programming”
  - Short Code (BINAC-1949) by John Mauchly
  - Speedcoding (IBM 701-1954) by John Backus
  - UNIVAC compiling systems A-0, A-1, and A-2 (1953)
    - Development team lead by Grace Hopper
    - Pseudocode expanded into machine code via macros
    - Precursor to assembly language

Grace Hopper

Fortran (1957)

- “FORmula TRANslation”
- Primary goal: execution speed
- Designed by John Backus at IBM
  - Practical alternative to assembly language
  - First widely-accepted compiled high-level language
- Original hardware: IBM 704 mainframe
  - Hardware floating-point implementation
- Very restrictive by today's standards
  - No block structure
  - One loop structure (DO)
  - Punchcard-restricted formatting
  - Implicit data types based on variable names



Punch card

IBM 704

Fortran (1957)

- Goal: half the efficiency of hand-written machine code
  - Largely successful! (likely partially due to Frances Allen's optimization work)
  - All modern high-performance compiler groups (Intel, Portland Group, etc.) maintain excellent Fortran support
- Highlights importance of a language's available compilers
  - Fortran remains the dominant language in high-performance computing, originally because of excellent compilers; more recently it is also because of the amount of existing legacy code
- Significant versions:
  - FORTRAN IV (1962): platform-independent
  - FORTRAN 77: block structures; broke backwards compatibility
  - Fortran 90: relaxed formatting guidelines
  - Fortran 2003: object-oriented support
  - Fortran 2008 / 2018: more parallel/concurrency features

Frances Allen

ALGOL (1960)

- “ALGOrithmic Language”
- Primary goal: independent general-purpose language
- Joint-effort design: ACM in the U.S. and GAMM in Germany
  - Generalization of Fortran features w/ several new contributions
  - First language syntax definition written in Backus-Naur Form
- Became widely used to describe algorithms in papers and publications
  - Influenced **many** languages over the subsequent decades
  - ALGOL 68 introduced user-defined data types built from primitives
- Never widely used in practice
  - Early BNF was difficult to understand
  - Hard to implement; too many confusing and overly-flexible constructs
  - No support from IBM (ALGOL was a competitor to Fortran)
- Question for the ages: what if ALGOL had "won?"

COBOL (1960)

- “COmmon Business-Oriented Language”
  - U.S. Department of Defense effort to create a portable language
  - Descendant of FLOW-MATIC data-processing language
    - Also developed by Grace Hopper
- Primary goal: easy for non-programmers to use
  - Resembled natural English and was very verbose
  - Used decimal arithmetic
  - First true implementation of records
- Gained tremendous momentum
  - At least seven major versions (latest in 2014)
- Lots of legacy code
  - Estimated 220+ billion lines in 2017!
  - Cautionary tale of IRS’s Individual Master File system: https://www.youtube.com/watch?v=qL5ut8o5pfs
- Little influence on subsequent PL design

```
OPEN INPUT sales, OUTPUT report-out
INITIATE sales-report

PERFORM UNTIL 1 <> 1
    READ sales
        AT END
            EXIT PERFORM
    END-READ

    VALIDATE sales-record
    IF valid-record
        GENERATE sales-on-day
    ELSE
        GENERATE invalid-sales
    END-IF
END-PERFORM

TERMINATE sales-report
CLOSE sales, report-out
```

Sample COBOL Program

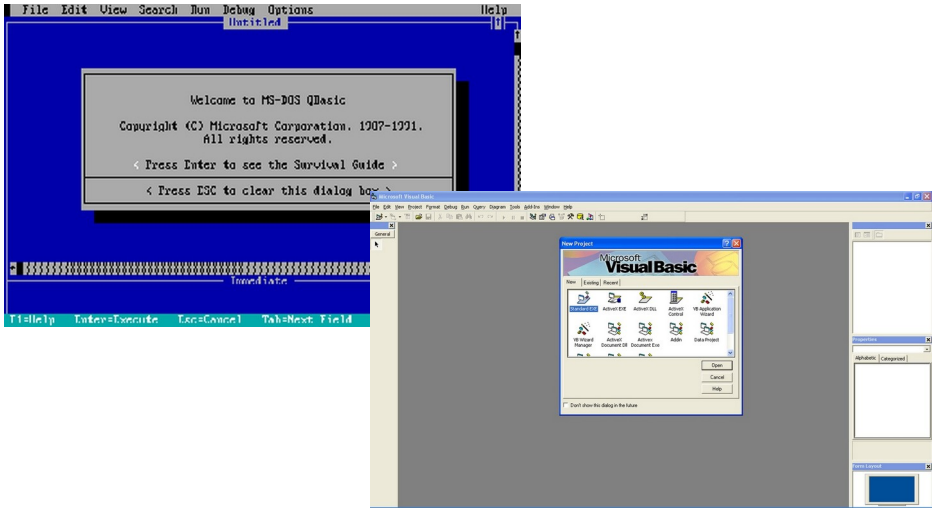
PL/I (1964)

- Programming Language One
- Primary goal: "Jack of all trades"
  - Combined best pieces of ALGOL, FORTRAN, and COBOL
  - Intended to replace them as well as Lisp and assembly
- Not widely used today
- However, it pioneered several lasting features:
  - Concurrent subprograms
  - Exception handling
  - Optional recursive subprograms
  - Pointers
  - Cross-sections (slices) of arrays

BASIC (1964)

- “Beginner's All-purpose Symbolic Instruction Code”
- Primary goal: simple and easy to learn
- Designed for non-science students
  - Emphasis on development time, rather than execution time
  - Prophetic, although not recognized as such at the time
- Descendants: QuickBasic and Visual Basic

```
REM SAY HELLO
10 PRINT "HELLO WORLD!"
20 GOTO 10
```



APL (1964)

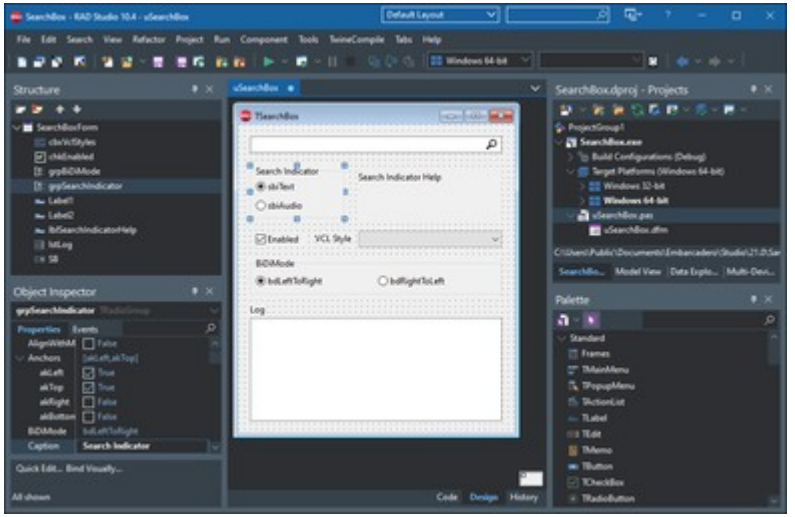
- "A Programming Language"
- Early dynamic language designed by Kenneth Iverson
- Originally designed to describe computer architectures
- Large number of operators
  - Specialized keyboard
  - Very concise code
  - Very unreadable code!
- Game of life:
  - `life←{↑1 ωv.∧3 4=+/,¯1 0 1∘.⊖¯1 0 1∘.⌽⊂ω}`

Early Object-Oriented Languages

- SIMULA (1967)
  - Primary goal: system simulation
  - Required restartable subprograms (coroutines)
  - Introduced a "class" construct for coroutine implementation
  - Precursor to object-oriented languages
- Smalltalk (1972)
  - Based on SIMULA
  - First major successful object-oriented language
  - Objects w/ state send messages to each other
  - Large influence on history of graphical user interfaces (GUIs)

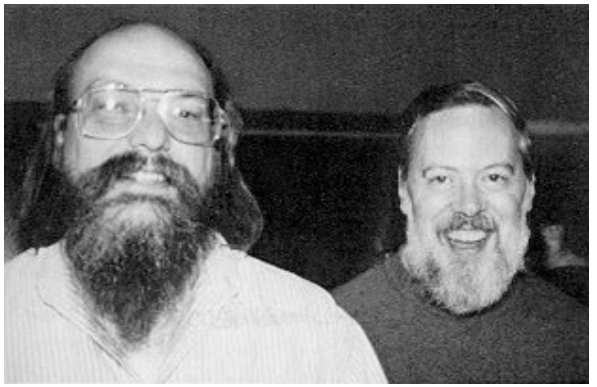
Pascal (1970)

- Based on ALGOL 60
- Designed by Niklaus Wirth
- Primary goal: simplicity and safety
- Widely used as an educational language
- Many extended dialects
  - Turbo Pascal
  - Object Pascal (Delphi)
  - Free Pascal



The Godfather: C (1972)

- Designed for systems programming
  - Influenced by ALGOL 68
  - Designed by Dennis Ritchie and others at Bell Labs
  - Based on a very similar (but untyped) language called B
- Tightly coupled with UNIX operating system
  - Close to the hardware
  - Simple but powerful constructs
  - Minimal type checking (both good and bad!)
  - Lack of true object-oriented capabilities
- Standards
  - Kernighan and Ritchie book (1978)
  - ANSI/ISO: C89/C90 (1989), C99 (1999), C11 (2011), C18 (2018)



Ken Thompson and Dennis Ritchie

Ada (1980)

- Originally designed for embedded systems
  - Another Department of Defense effort
  - Monolithic design process (1974-1980)
  - Named after Ada Lovelace
- Primary goal: good PL principles
  - Enforce software engineering best practices
  - Largely succeeded
- Large, complex, and hard to implement
  - Design published in 1980 and standardized in 1983
  - The first useful compiler was not finished until 1985
- Could not compete with C
  - Many people conjecture that software today would be much safer in general if Ada (and its principles) had gained more widespread support

Augusta Ada Lovelace

C++ (1985)

- Extended C with object-oriented features
  - Bjarne Stroustrup at Bell Labs
- Primary goal: speed and flexibility w/ low-cost abstractions
  - Originally "C with classes" preprocessor
  - Adds objects, templates, exceptions, and **lots** more
  - Few programmers use every feature of C++
- Became tremendously popular as OOP flourished
  - ANSI standardized in 1998
  - “Right place, right time”
  - Even influenced post-1989 versions of C
  - Major revision in 2011 (C++11)
  - Later revisions in 2014, 2017, and 2020

Bjarne Stroustrup

Java (1995)

- Originally designed for embedded applications
- Primary goal: reliability and portability
- Based on C++ but features were simplified and reduced
  - Eliminated pointers and added automatic garbage collection
  - Cleaned up templates (now "generics") and exceptions
  - Much stronger type checking
  - Often used for CS education
- Programs run on the Java Virtual Machine
  - Core runtime system that must be implemented for every new architecture
  - Source code is compiled to "byte code," which is interpreted by the JVM
  - Individual programs are very portable
- Inspired Groovy (2003), Scala (2004), and Kotlin (2011)

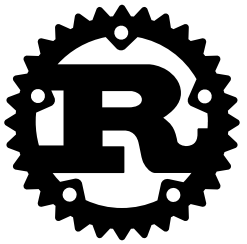
.NET Languages (starting 2002)

- Common Language Infrastructure (CLI)
  - Language-neutral runtime platform
  - Common Intermediate Language (CIL) and Portable Executable (PE) format
- C# is a descendant of C++ and Java
  - Based on Java
  - Re-introduced some C++ features (but not multiple inheritance!)
- VB.NET is a descendant of BASIC (via Visual Basic)
  - Emphasis on writability and business applications
- J# and JScript.NET are transitional languages for Java/Javascript users
- F# is a multi-paradigm (including functional) language
- ASP.NET for server-side web apps, Q# for quantum computing
- If you're in the Windows world, .NET is great
  - Includes excellent developer support via the Visual Studio suite

Other Notable C/C++ Descendants

- Objective C (1983) and Swift (2014)
  - Descendants of C and Smalltalk; primarily used today by Apple
- D (2006)
- Go (2009)
  - Developed at Google by Ken Thompson and others

- Rust (2012)
  - Multi-paradigm client/server and systems language from Mozilla
- Common theme: a “modern” redesign of C/C++
  - Without the major problems and headaches
  - Added features for greater safety, concurrency, etc.
  - None have yet succeeded in displacing C/C++



Dynamic Scripting Languages

- Perl (1987) – Larry Wall
  - Originally designed for text processing
  - Powerful but ugly: the “Swiss Army chainsaw” of PL
  - Widely used across many domains, especially CGI programming
- Python (1991) – Guido van Rossum
  - Strong design philosophies
  - Large standard library
  - Python 3.0 (2008) is backwards-incompatible

- Ruby (1995) – Yukihiro Matsumoto
  - Designed for "productivity and fun"
  - Pure object-oriented multi-paradigm language
  - Strong self-inspection features ("reflection")

Dynamic Scripting Languages

- Javascript (1995) – Brendan Eich at Netscape
  - Embedded web browser and document programming
  - **Similar to Java in syntax, but otherwise very different!**
  - Widely used, but has been a source of many security flaws
  - Many popular extensions: Node.js, JSON, jQuery, etc.
- Others: AppleScript, Bash, Lua, PHP, R, TCL, Typescript, VBScript
- General themes of dynamic scripting languages:
  - Interpreted (sometimes compiled for speed)
  - Powerful, expressive, and flexible syntax
  - Dynamic and/or "duck" typing
  - Automatic memory management
  - Anyone who doesn't use your preferred scripting language is clearly ~~wrong~~ ignorant and must be ~~converted~~ educated

Modern Fortran alternatives

- Chapel (2009)
  - Compiled language for high-performance computing
  - Balances expressivity, performance, and modern language features
  - Multiple levels of concurrency abstractions
  - Support for implicit distributed memory programming
- Julia (2012)
  - Combines ideas from Python, MATLAB, R, and Fortran
  - Mantra: "*vectorize when it feels right*"
  - Core is implemented in C/C++, JIT-compiled to native machine code
- Project Jupyter (2015)
  - **Ju**lia, **Pyt**hon, and **R**
  - Jupyter Notebook: web-based REPL
    - Ordered list of “cells” containing code, text, or other media

Programming Paradigms

- Procedural
  - Includes all previously-discussed languages
  - And probably every language you saw before 430
  - It's definitely here to stay (at least for the foreseeable future)
  - But it's not the only paradigm
- Functional
  - LISP, Scheme, and descendants (including **Haskell**)
- Declarative / Logic
  - **Prolog** and descendants

LISP (1959)

- LISt Processing language
  - John McCarthy at MIT
- Functional language based on mathematics (lambda calculus)
  - No variables or global state
    - No side effects! This makes reasoning about program correctness much simpler and more powerful.
  - All computation involves applying functions to inputs
  - Iteration via recursion
  - Data types: atom and list (atom + list)
  - Symbolic computation
- Used primarily for AI research

```
(defun factorial (n)
  (if (= n 0) 1
      (* n (factorial (- n 1)))))
```

Descendants of LISP

- Common Lisp (1984/94)
  - Consolidation of many LISP variants
  - Multi-paradigm (supports procedural programming as well)
- Scheme (1970s)
  - Designed by Guy Steele and Gerald Sussman
  - Simplification of LISP; often used as a teaching language
- ML - MetaLanguage (1973)
  - Strongly-typed proof language designed by Robin Milner
  - Later extended at INRIA (France) to Caml and OCaml
- Haskell (1990)
  - Named after influential logician Haskell Curry
  - Purely functional language w/ strong typing and lazy evaluation

Prolog (1972)

- Declarative / Logic programming language
- Based on first-order predicate logic
  - Built-in goal-directed inference engine
  - Given facts and implications
  - Uses inference to infer the truth of queries
- Drawbacks
  - Can be difficult to understand
  - Solutions are often inefficient and/or of limited usefulness

```
% read as “sally has mother alice”
mother(sally, alice).

father(sally, tom).
father(erica, tom).
father(tom, mike).

% parent(X, Y) is read as “X has parent Y”
parent(X, Y)  :- father(X, Y).
parent(X, Y)  :- mother(X, Y).
sibling(X, Y) :- parent(X, Z), parent(Y, Z).
```

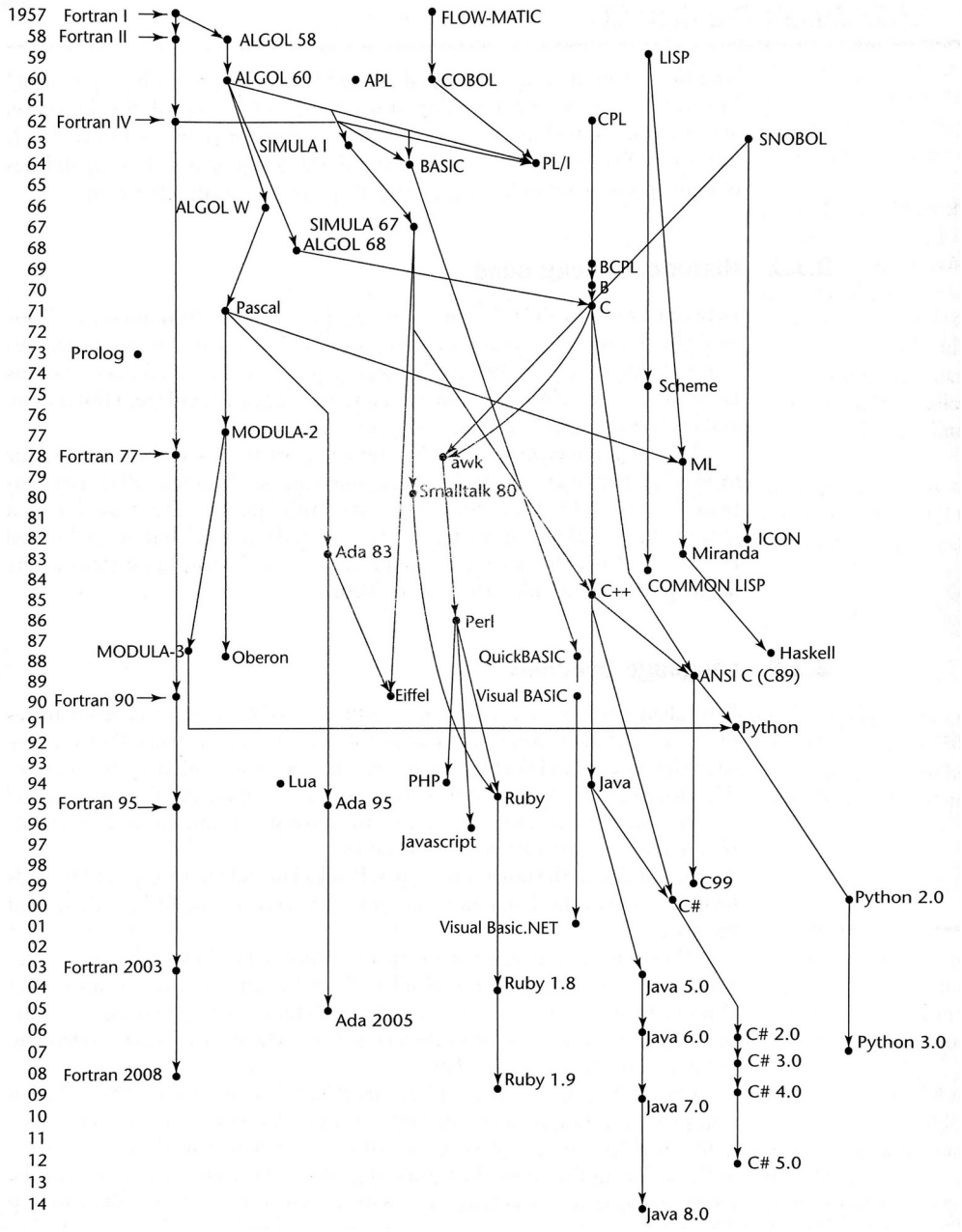
```
?- sibling(sally, erica).
Yes

?- parent(erica, alice).
No
```

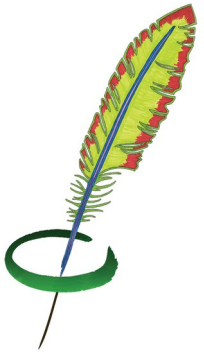
Functional and Logic Paradigms

- Functional languages are becoming more mainstream
  - Concepts are extremely useful
  - Gaining popularity as software becomes more complex and concurrency becomes more important
  - Many procedural languages are adding functional features (including heavy-hitters like C# and Java)
  - Good tool to know
- Logic languages are more of a niche
  - Rarely used in practice
  - Useful for pattern matching w/ rules (e.g., IBM Watson)
  - Curry-Howard isomorphism: “programs are proofs”
  - Good tool to be aware of

Language Family Tree



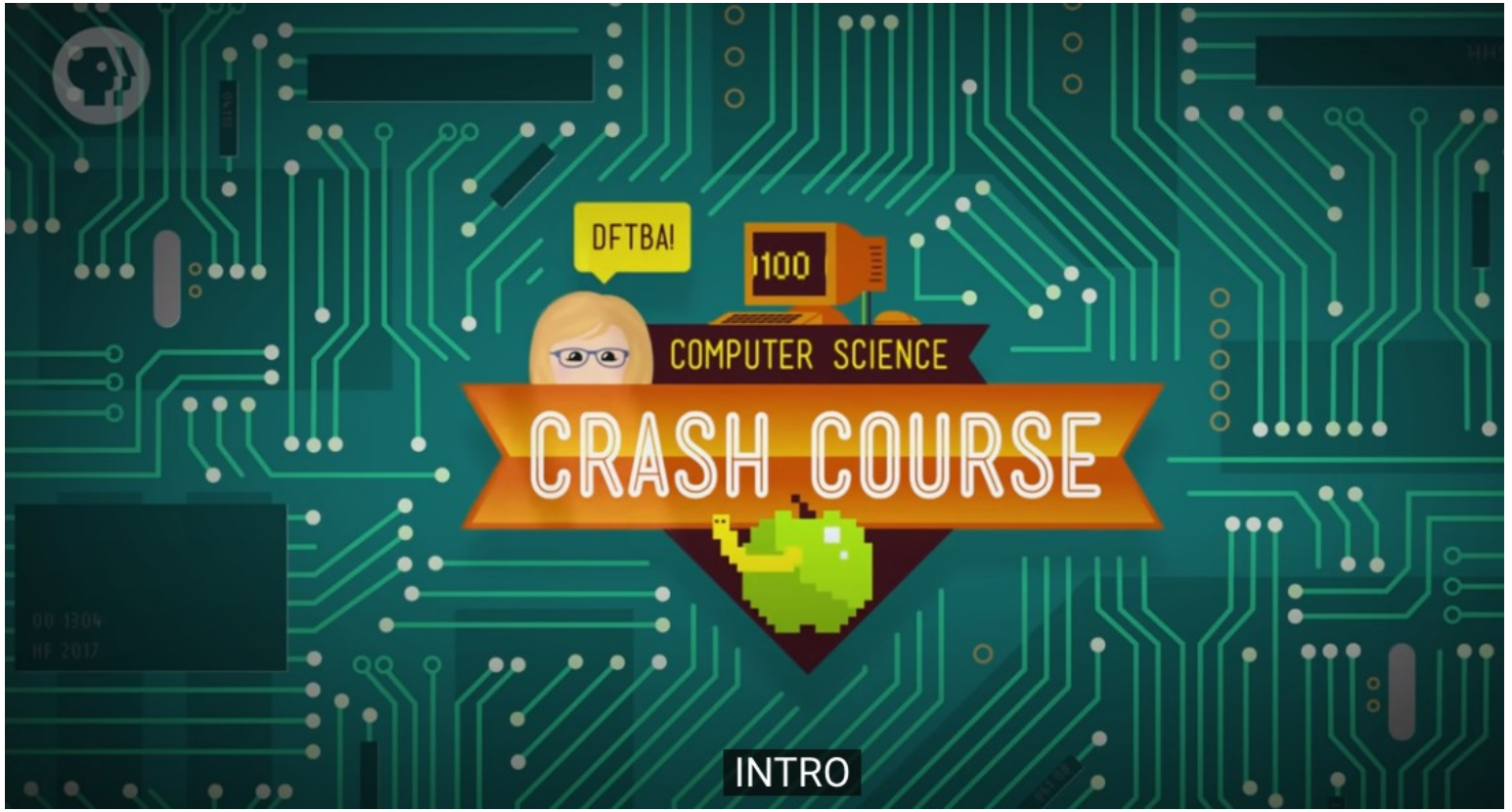
“History of PL” Conferences



- HOPL-I (1978)
  - Keynote by then-Captain Grace Hopper
  - FORTRAN, ALGOL, LISP, COBOL, SIMULA, BASIC, PL/I, APL, and others
  - https://dl.acm.org/doi/book/10.1145/800025
- HOPL-II (1993)
  - Prolog, Smalltalk, C, Forth, C++, Ada, Pascal, and others
  - https://dl.acm.org/doi/proceedings/10.1145/154766
- HOPL-III (2007)
  - “50 in 50” talk by Guy Steele and Richard Gabriel
  - Lua, C++, Erlang, “Rise and Fall of High Performance Fortran”, ZPL, Haskell, and others
  - https://dl.acm.org/doi/proceedings/10.1145/1238844
- HOPL-IV (2020)
  - APL, **C++**, Fortran, D, Emacs Lisp, F#, Groovy, **Javascript**, LabVIEW, Logo, MATLAB, Objective-C, R, Smalltalk, Standard ML, Verilog, and others
  - https://dl.acm.org/toc/pacmpl/2020/4/HOPL

Crash Course Computer Science

- Video #11: “The First Programming Languages”



https://www.youtube.com/watch?v=RU1u-js7db8

Your presentations

Generated by `freewordcloudgenerator.com`

“There are only two kinds of programming languages; those people always [complain] about and those nobody uses.”

– Bjarne Stroustrup