CMSC 330: Organization of Programming Languages

A Brief History of Programming Languages

CMSC330 Fall 2020

Babylon

- Founded roughly 4000 years ago
  - Located near the Euphrates River, 56 miles south of Baghdad, Iraq
- Historically influential in ancient western world
- Cuneiform writing system, written on clay tablets
  - Some of those tablets survive to this day
  - Those from Hammurabi dynasty (1800-1600 BC) include mathematical calculations
    - (Also known for Code of Hammurabi, an early legal code)

A Babylonian Algorithm

A [rectangular] cistern.
The height is 3, 20, and a volume of 27, 46, 40 has been excavated.
The length exceeds the width by 50.
You should take the reciprocal of the height, 3, 20, obtaining 18.
Multiply this by the volume, 27, 46, 40, obtaining 8, 20.
Take half of 50 and square it, obtaining 10, 25.
Add 8, 20, and you get 8, 30, 25
The square root is 2, 55.
Make two copies of this, adding [25] to the one and subtracting from the other.
You find that 3, 20 [i.e., 3 1/3] is the length and 2, 30 [i.e., 2 1/2] is the width.
This is the procedure.

– Donald E. Knuth, *Ancient Babylonian Algorithms*, CACM July 1972

The number n, m represents $n*(60^k) + m*(60^{k-1})$ for some k

More About Algorithms

- Euclid's Algorithm (Alexandria, Egypt, 300 BC)
  - Appeared in *Elements*
    - *See http://aleph0.clarku.edu/~djoyce/elements/bookVII/propVII2.html*
  - Computes gcd of two integers
   ```
    let rec gcd a b =
        if b = 0 then a else gcd b (a mod b)
    ```

- Al-Khwarizmi (Baghdad, Iraq, 780-850 AD)
  - *Al-Khwarizmi Concerning the Hindu Art of Reckoning*
  - Translated into Latin (in 12th century?)
    - Author's name rendered in Latin as *algoritmi*
    - Thus the word *algorithm*

# Antikythera Mechanism



- Mechanical computer designed to calculate astronomical positions
- Discovered in 1900-1901
  - But no one knew what it was until much later!
- Estimated construction in 150-100 BC

# Charles Babbage (1791–1871)
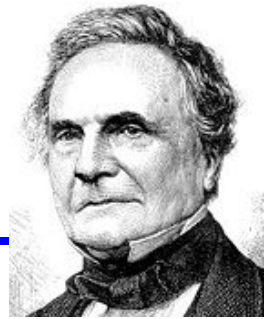
- British mathematician & mechanical engineer
- Invented concept of a programmable computer
  - Partially built a Difference Engine

- A mechanical calculator for computing polynomials
- Uses finite difference method to avoid multiplication & division
- Never completely finished
- Model built in 1991 at the London Science Museum
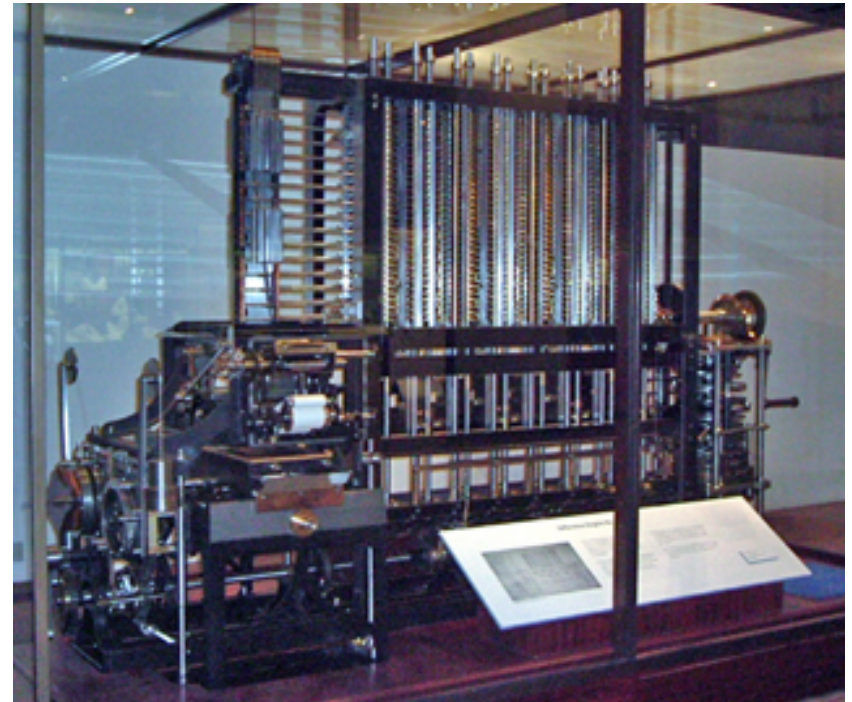
# The Analytical Engine (1837)

- Babbage described plans for an Analytical Engine
  - Digital, programmable using punch cards
  - Included branching, looping, arithmetic, and storage
  - Memory would store 1000 numbers w/ 500 digits each
  - Language similar to assembly language
  - Powered by steam engine
- Partially constructed by 1910
  - Used to compute a list of multiples of $\pi$
- If built, would have been 1st Turing-complete computation device

# Ada Lovelace (1815-52)



- British mathematician
  - Corresponded with Babbage from 1833 onward
  - In 1843, translated Italian mathematician L. Menabrea's memoir on Babbage's proposed Analytical Engine
  - Appended notes on how to program the Analytical Engine to calculate Bernoulli numbers
  - Recognized by historians as 1st published program
  - Making her world's 1st programmer
- Ada programming language (1983)
  - Imperative, object-oriented language based on Pascal
  - Named in her honor

# Ada Lovelace (1815-52)

- "The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform."

- She speculated that the Engine 'might act upon other things besides number... the Engine might compose elaborate and scientific pieces of music of any degree of complexity or extent'.

- She was the first to express the potential for computers outside mathematics.

# Alonzo Church (1903-1995)



- Mathematician at Princeton Univ.

- Three key contributions:
  - The lambda calculus (lectures in 1936, publ. 1941)
  - Church's Thesis
    - All effective computation is expressed by recursive (decidable) functions
  - Church's Theorem
    - First order logic is undecidable

# Alan Turing (1912 - 1954)
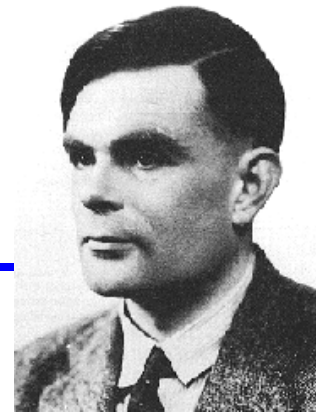
- ## The father of modern computer science

  – Dissertation work advised by Church at Princeton

  – Formulated the Turing machine (~1936)

  A general abstract computer → DFA + infinite 1D tape

  - Σ – A finite alphabet

  - Q – a set of states

  - s ϵ Q – A start state

  - F ⊆ Q – The final states

  - δ : Q × Σ → Q × Σ × {L, R}

    – If δ(q, a) = (q', a', d), then if we're in state q and see a on the tape, then replace it by a', move to state q', and move the position of the tape either left or right

  – A formal definition of a computable algorithm

# Other Early Computers

- ABC (1939-1942)
  - Atanasoff and Berry Computer, at Iowa State Univ.
  - First electronic digital computer
    - As decided by a judge in 1973!  (Invalidated ENIAC patent)

- Z3 (1945)
  - Konrad Zuse, essentially isolated from everyone else
  - Used Plankalkül, a sophisticated programming lang.
    - But no one knew about his results, so not influential

- Harvard Mark I (1944)
  - Aiken, IBM
  - Electronic, used relays

# Other Early Computers (cont.)

- ENIAC (1946)

  - Electronic Numerical Integrator and Computer

  - Developed by Eckert and Mauchly at UPenn

  - Electronic, general purposes

  - Used vacuum tubes

  - For 30 years considered the "first" electronic computer

    - Until court case gave honor to ABC

    - Supposedly Eckert and Mauchley overheard Atanasoff discussing designs for ABC

# ENIAC early photo
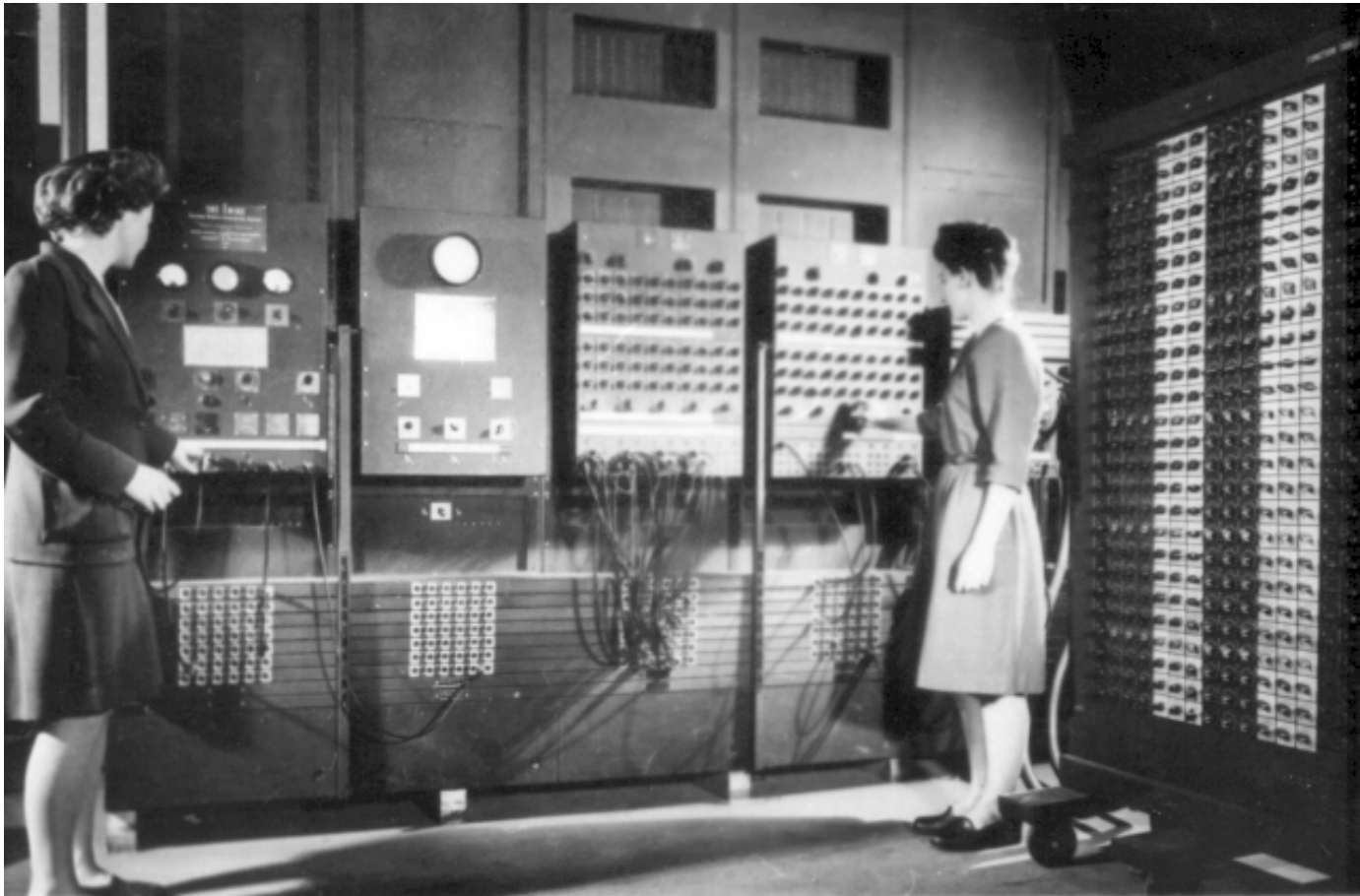


Women not named in the picture – they are programmers

# The ENIAC Six:
# Women who Programmed the ENIAC

# The First Programming Languages

- Early computers could be "programmed" by rewiring them for specific applications
  - Tedious, error prone

- John von Neumann (1903-1957)
  - Three CS contributions (famous for lots of other stuff)
    - von Neumann machine – the way computers are built today
      - A *stored program architecture*
        » Program stored in memory as data, so can be modified
      - (Unclear that he actually invented this...)
    - "Conditional control transfer" – if and for statements
      - Allows for reusable code, like subroutines
    - Merge sort algorithm

# Pseudocodes (Assembly Interpreter)

- ## Short Code (1949)
  - John Mauchly
  - Interpreted instructions
    - E.g., X0 = sqrt(abs(Y0)) becomes 00 X0 03 20 06 Y0
      - 06 = abs, 20 = sqrt, 03 = assignment
  - But needed to translate by hand

- ## A-0 Compiler (1951; Grace Murray Hopper)
  - Translated symbolic code into machine code
    - Sounds like an assembler...
  - Assigned numbers to routines stored on tape
    - Which would then be retrieved and put in memory

Eckert and Mauchly

# FORTRAN (1954 - 1957)

- FORmula TRANslator
- Developed at IBM by John Backus et al
  - Aimed at scientific computation
  - Computers slow, small, unreliable
    - So FORTRAN needed to produce efficient code
- Features (FORTRAN I)
  - Variable names (up to 6 chars)
  - Loops and Arithmetic Conditionals
    - IF ($E$) 100, 200, 300
      - Go to 100 if $E$ is negative, 200 if zero, 300 if positive
  - Formatted I/O
  - Subroutines
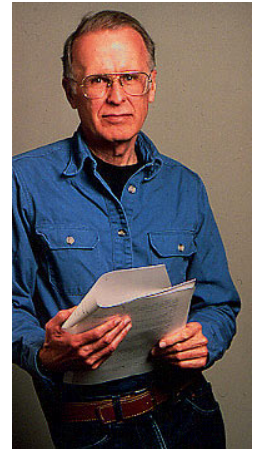
# Writing FORTRAN Programs

- Programs originally entered on punch cards
  - Note bevels on top-left corner for orientation
  - First five columns for comment mark or statement number
  - Each column represents one character
  - Letter: 2 punches: A=12,1 B=12,2 …, Z=0,9
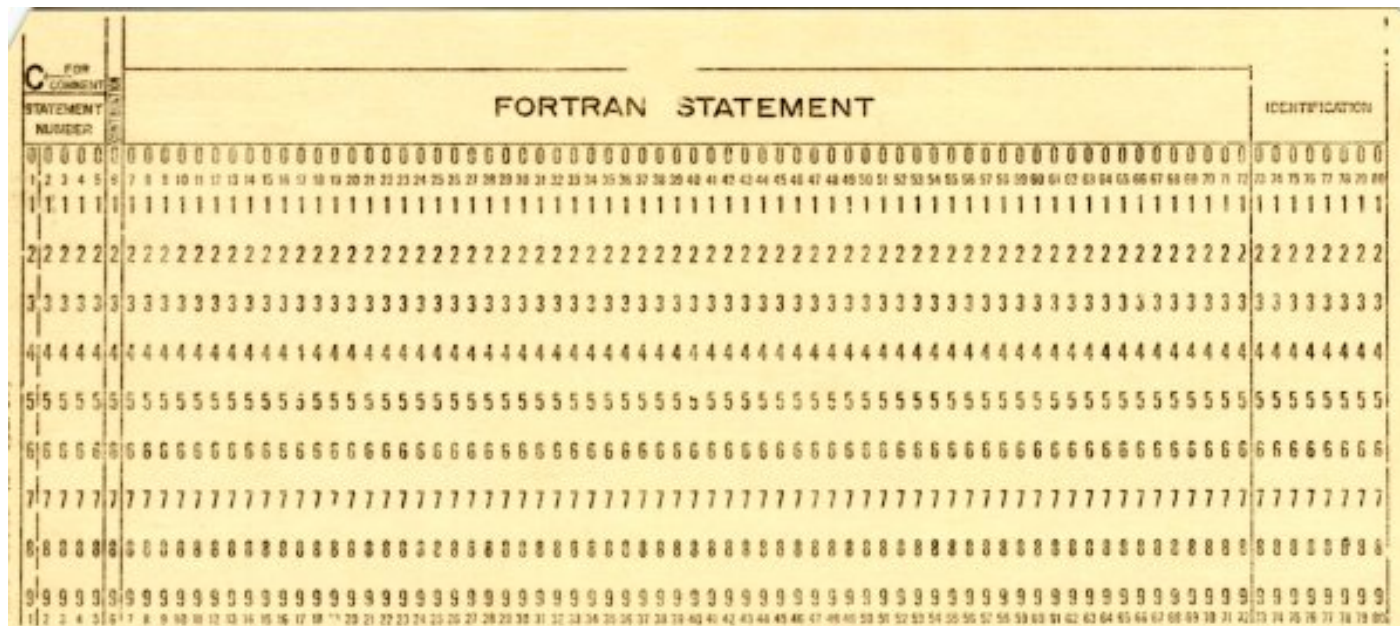
# Punch Card Programming

- Not interactive!
  - Feed the deck into the machine
    - Or give it to someone to put in
  - Eventually get back printout with code and output
    - Could take a couple of hours if machine busy
    - Student jobs typically took overnight to run
      - Only to find a syntax error!

- Long test-debug cycle
  - Debugging by hand critical to not wasting time
    - Don't want to wait several hours to find you made a typo

- What happens if you drop your deck of cards?
  - Could put sequence number in corner for ordering
  - Hard to maintain this as you keep modifying program

# Example (FORTRAN 77)

C = "comment"

All-caps

For loop; I goes from 2 to 12 in increments of 1

Cols 1-6 for comment or stmt label

```
C A PROGRAM TO COMPUTE MULTIPLICATION TABLES
      PROGRAM TABLES
      DO 20 I = 2,12
      PRINT *,I,' TIMES TABLE'
      DO 10 J = 1,12
10       PRINT *,I,' TIMES',J,' IS',I*J
20       CONTINUE
      END
```

Source: University of Strathclyde Computer Centre, Glasgow, Scotland

End of program

FORTRAN Parsing Example

- Blanks don't matter in FORTRAN
  - And identifiers can have numbers in them

- Consider parsing the following two statements

```
DO 20 I = 2,12
DO 20 I = 2.12
```

  - The first is a loop
  - The second is an assignment of 2.12 to "DO20I"
  - Notice that we need many characters of lookahead to tell the difference between the two for a parser
  - The statement: DO 3 I = 1.3 may have caused the failure of Mariner I to Venus.* (FORTRAN assumed DO3I= 1.3, which is legal, but not the loop that was desired.)
    - *This story seems to be apocryphal, but is printed in many textbooks

FORTRAN Today

- Success of FORTRAN led to modern compilers
- Up to FORTRAN 95
  - Modern version of the original language
  - Includes pointers, recursion, more type checking, dynamic allocation, modules, etc.
- Only (?) major language with column-major arrays
- Still popular for scientific computing applications
  - FORTRAN compilers produce good code
  - Language has an efficient array structure
    - C has pointers, which can hurt optimization
      - FORTRAN has arrays, but compiler can assume that if x and y are arrays then x and y are unaliased
    - Java interpreted or JIT'd, and uses dynamic dispatch heavily

COBOL (1959)

- COmmon Business Oriented Language
  - Project led by Hopper (again!)
- Design goals
  - Look like simple English (but doesn't read like it!)
  - Easy to use for a broad base
  - Notice: not aimed at scientific computing
    - Aimed at business computing instead, very successfully
- Key features
  - Macros
  - Records
  - Long names (up to 30 chars), with hyphen

COBOL Example (Part 1)

Program data
(variables)

Single-digit number

Initial value

```
        $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  Iteration-If.
AUTHOR.  Michael Coughlan.


DATA DIVISION.
WORKING-STORAGE SECTION.
01  Num1              PIC 9  VALUE ZEROS.
01  Num2              PIC 9  VALUE ZEROS.
01  Result            PIC 99 VALUE ZEROS.
01  Operator          PIC X  VALUE SPACE.
```

Source: http://www.csis.ul.ie/COBOL/examples/conditn/IterIf.htm

Level number; can be used
to express hierarchy (records)

The source of Y2K bugs

Character

COBOL Example (Part 2)

```
PROCEDURE DIVISION.          Iteration
Calculator.
    PERFORM 3 TIMES
        DISPLAY "Enter First Number     : " WITH NO ADVANCING
        ACCEPT Num1
        DISPLAY "Enter Second Number    : " WITH NO ADVANCING
        ACCEPT Num2
        DISPLAY "Enter operator (+ or *) : " WITH NO ADVANCING
        ACCEPT Operator
        IF Operator = "+" THEN
           ADD Num1, Num2 GIVING Result
        END-IF
        IF Operator = "*" THEN
           MULTIPLY Num1 BY Num2 GIVING Result
        END-IF
        DISPLAY "Result is = ", Result
    END-PERFORM.
    STOP RUN.
```

Data Layout

- Variables have fixed position in memory
  - Same with early FORTRAN compilers

- Advantages?
  - Compilers easier to write
  - Efficient at runtime

- Disadvantages?
  - No dynamic memory alloc (i.e., no data structures)
  - No recursive functions

Skynet Runs Cobol

CMSC330 Spring 2020
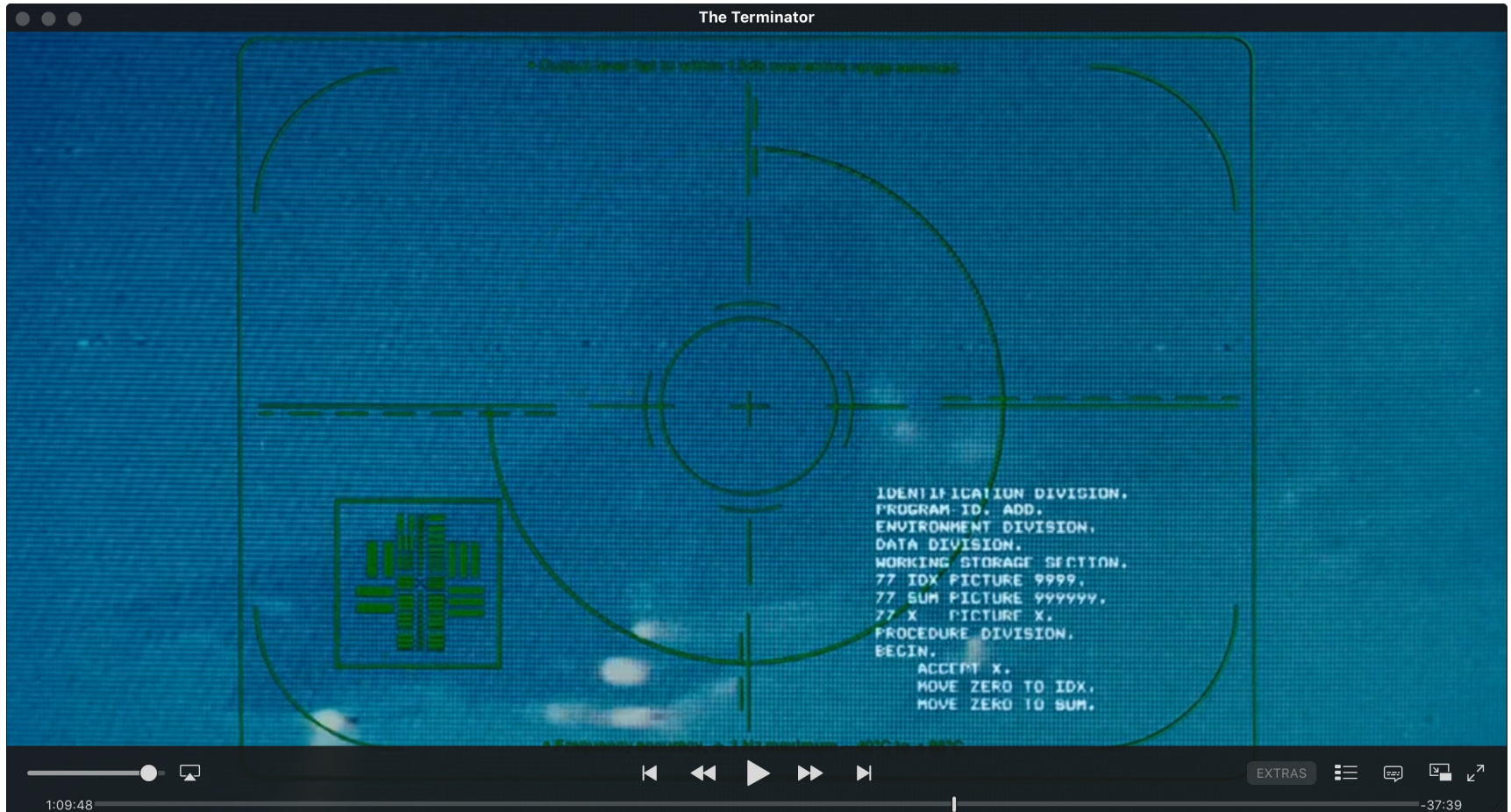
COBOL Today

- Was a DoD requirement at one time
  - Important element of its success

- Still used today
  - #36 on IEEE list
    - http://spectrum.ieee.org/static/interactive-the-top-programming-languages
  - Legacy mainframe applications
  - New standard in 2002
  - Language has been updated for new features
    - Object-oriented?!
    - Unicode support
    - XML support

LISP (1958)

- LISt Processing

- Developed by John McCarthy at MIT

  – Designed for AI research

- Key ideas:

  – Symbolic expressions instead of numbers

  – Lists, lists, lists

  – Functions, functions, functions

    - Compose simpler functions to form more complex functions

    - Recursion

  – Garbage collection (mark/sweep)

LISP Code

```
(defun factorial (n)
        (cond ((zerop n) 1)
                (t (times n (factorial (sub1 n)))))
```

- Implemented on IBM 704 machine

  – Machine word was 36 bits

    • Two 15-bit parts, "address" and "decrement," distinguished
    • car = "Contents of the Address part of Register"
    • cdr = "Contents of the Decrement part of Register"

- Invented maplist function

  – Same as map function in OCaml

- Used lambda notation of Church for functions

LISP Code as Data

- Notice that LISP programs are S-expressions
  - Which represent lists

- So LISP programs can easily manipulate LISP programs
  - Just do list operations to put programs together
  - Probably the first high-level language with this feature

LISP Machines (Later Development)

- LISP is a fairly high-level language
  - Small pieces of LISP code can require a fair amount of time to execute
  - Running a LISP program could kill a timeshared machine

- Idea: design a machine specifically for LISP
  - Well-known examples are from Symbolics
    - Everything on the machine written in LISP
  - Killed by the PC revolution

Scheme (1975)

Steele and Sussman

- Dialect of LISP

- Developed buy Guy L. Steele and Gerald Jay Sussman
  - Guy Steele was heavily involved in Java

- Goal: Minimalist language
  - Much smaller than full LISP
  - First dialect of LISP to include static scoping
  - Used by people experimenting with languages

LISP Today

- LISP still used in AI programming
    - Common LISP is standard version

- Scheme still popular in academics
    - Good vehicle for teaching introductory programming
        - The TeachScheme! Project
        - Origins of CMSC131A, piloting at UMD
    - Racket language started from Scheme

- Clojure
    - Niche LISP dialect focusing on concurrent programming

Algol (1958)

- ALGOrithmic Language
  - Designed to be a universal language
  - For scientific computations
- Never that popular, but extremely important
  - Led to Pascal, C, C++, and Java
    - "Algol-like" languages
  - Had formal grammar (Backus-Naur Form or BNF)
  - Algol 60 added block structures for scoping and conditionals
  - Imperative language with recursive functions

Example Code

```
procedure Absmax(a) Size:(n, m) Result:(y)
                    Subscripts:(i, k);
    value n, m; array a;
    integer n, m, i, k; real y;
comment The absolute greatest element of the
matrix a, of size n by m is transferred to
y, and the subscripts of this element to i
and k;
begin integer p, q;
    y := 0; i := k := 1;
    for p:=1 step 1 until n do
    for q:=1 step 1 until m do
        if abs(a[p, q]) > y then
            begin y := abs(a[p, q]);
            i := p; k := q
            end
end Absmax
```

Algol 68

- ## Successor to Algol 60
  - But bloated and hard to use
  - And very hard to compile
    - E.g., variable names can include blanks!

- ## Included many important ideas
  - User-defined types
  - Code blocks that return the value of the last expr
  - Struct and union
  - Parallel processing (in the language)

Example Code

User-defined types

Identifier w/blank

Malloc

Field access

```
BEGIN MODE NODE = STRUCT (INT k, TREE smaller, larger),
        TREE = REF NODE;

    TREE empty tree = NIL;

    PROC add = (REF TREE root, INT k) VOID:
        IF root IS empty tree
        THEN root := HEAP NODE := (k, NIL, NIL)
        ELSE IF k < k OF root
                THEN add (smaller OF root, k)
                ELSE add (larger OF root, k)
                FI
        FI;
END
```

Source: http://www.xs4all.nl/~jmvdveer/algol68g-mk8/doc/examples/quicksort.a68.html

Algol Discussion

- Good points:
  - Standard for writing algorithmic pseudocode
  - First machine-independent language
  - C, C++, Java, etc. all based on it
  - Used BNF to describe language syntax

- Bad points:
  - Never widely used; Some success in Europe
  - Hard to implement
  - FORTRAN much more popular
  - Not supported by IBM

APL (early 1960s)

- A Programming Language
  - Developed by Kenneth Iverson at Harvard
  - Goal is a very concise notation for mathematics
    - Focuses on array manipulation, interactive environment
  - Was very popular
    - In 1969, 500 attendees at APL conference
    - Still some devotees

- Notable features
  - Programs evaluated right-to-left, No precedence of operators
  - Fast execution of array operations
  - Required special keyboard to enter programs

Example Code

$$(\sim R \; \epsilon \; R \; o.\times \; R) \; / \; R \leftarrow 1 \downarrow \iota R$$

- Finds prime numbers from 1 to R
  - ιR creates vector containing 1..R
  - $1\downarrow$ drops first element
  - $R\leftarrow$ assigns result to R
  - $R \; o.\times \; R$ computes (outer) product 2..R $\times$ 2..R
  - $R \; \epsilon$ produces vector the same size as R with 1 in position i if i is in $R \; o.\times \; R$
    - I.e., if there is a product of two numbers equal to i
  - $\sim$ negates the resulting vector; call it S
  - / returns a vector of items in R that have 1's in same pos in S

PL/I (1964)

- One language to replace FORTRAN and COBOL
    - Went along with System/360, one mainframe to replace mainframes for science, business
    - Syntax based on ALGOL
- Famous parsing problems

    IF THEN THEN THEN = ELSE; ELSE ELSE = THEN

    - No reserved words. Keywords could be used as variable names
- In the end, not that successful
    - Compilers hard to write
        - Not just because of parsing; language is complex
        - Inconsistency. 1/3+10 is an illegal expression. (Overflow!)
    - Looked down on by both camps (science, business)

BASIC (1964)

- Beginner's All purpose Symbolic Instruction Code
  - John Kemeny and Thomas Kurtz (Dartmouth)
    - Kemeny's PhD advised by Church
    - Kemeny also developed the first time sharing system
      - Multiple users could access shared mainframe as if they were the only user

- Goals of BASIC:
  - Easy to use for beginners
  - Stepping-stone to more powerful languages
  - User time is more important than computer time
  - First program run May 1, 1964 at 4:00 am

Example (Applesoft BASIC)

Variables

Lines numbered

Comment

Loop

Goto common

```
10  INPUT "What is your name: "; U$
20  PRINT "Hello "; U$
25  REM
30  INPUT "How many stars do you want: "; N
35  S$ = ""
40  FOR I = 1 TO N
50  S$ = S$ + "*"
55  NEXT I
60  PRINT S$
65  REM
70  INPUT "Do you want more stars? "; A$
80  IF LEN(A$) = 0 THEN GOTO 70
90  A$ = LEFT$(A$, 1)
100 IF (A$ = "Y") OR (A$ = "y") THEN GOTO 30
110 PRINT "Goodbye ";
120 FOR I = 1 TO 200
130 PRINT U$; " ";
140 NEXT I
150 PRINT
```

Source: http://en.wikipedia.org/wiki/BASIC_programming_language

BASIC Today

- Still an extremely popular language
  - Microsoft Visual Basic convenient language for writing basic Windows applications
    - Lots of libraries available
    - Includes development environment for gui apps

- Modern dialects of BASIC are more structured
  - Eliminate most or all line numbers
  - Discourage use of "goto"
  - BASIC today is a one word oxymoron

Pascal (1971)

- Developed by Niklaus Wirth
  - A response to complaints about Algol 68

- Teaching tool; not meant for wide adoption
  - Was popular for about 20 years
  - Best features of COBOL, FORTRAN, and ALGOL
  - And used lowercase!
    - Lowercase not introduced into ASCII until 1967
    - Even into 80's some mainframes were 6-bit, no lowercase

- Pointers improved

- Case statement

- Type declarations

Interesting Pascal Features

- Enumeration and subrange types

```
type day = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
type weekday = Mon..Fri;
type day_of_month = 1..31;
```

  – Key features:
  - Safe; values will always be within range (compare to C)
    – May require dynamic checks

- Array types with arbitrary ranges

```
var hours : array[Mon..Fri] of 0..24;
var M : array[Mon..Fri] of array[char] of real;
for day := Mon to Fri do
   M[day]['e'] := 0.0001;
```

Interesting Pascal Features (cont.)

- ## Set types

```
var S, T : set of 1..10;
S := [1, 2, 3, 5, 7];
T := [1..6];
U := S + T;    { set union }
if 6 in S * T then ... { set intersection }
```

  – (Note comments in {}'s)

Pascal Today

- Kernighan's Why Pascal is Not My Favorite Programming Language
    - No polymorphism on types, including arrays
    - No separate compilation (although every implementation created a module structure)
    - No short-circuiting && and ||
    - Weak run-time environment
    - No escape from type system
- Most of these problems fixed
    - But differently in different compilers
    - Not much used today

C (1972)

Thompson and Ritchie

- Dennis Ritchie at Bell Labs
  - Ancestors B and BCPL
  - Designed to build Unix
    - OSes previously written in Assembly, so platform dependent

- Two key features
  - Arrays and pointers closely related
    - `int *p` and `int p[]` are the same
    - Consequence of low-level view of memory
  - Type system lets you use values at any time
    - Type casts are necessary
    - Early compilers didn't complain about all sorts of things
      - Like assigning integers to pointers or vice-versa

Simula (1965)

- Developed at Norwegian Computing Center
  - By Ole-Johan Dahl and Kristen Nygaard
  - Goal was to simulate complex systems
  - Later used as a general purpose language
- Key features
  - Classes and objects
  - Inheritence and subclassing
  - Pointer to objects
  - Call by reference
  - Garbage collection
  - Concurrency via coroutines

Example

Parameter types

null pointer

```
class Point(x,y); real x,y;
  begin
    boolean procedure equals(p); ref(Point) p;
    if p =/= none then
      equals := abs(x - p.x) + abs(y - p.y) < 0.00001
    real procedure distance(p); ref(Point) p;
      if p == none then error else
        distance := sqrt((x - p.x)**2 + (y - p.y)**2);
end ***Point***


p :- new Point(1.0, 2.5);
q :- new Point(2.0, 3.5);
if p.distance(q) > 2 then ...
```

Return value

Field access

Pointer assignment

Source: http://www.stanford.edu/class/cs242/slides/2004/simula-smalltalk.pdf

Smalltalk (Early 1970s)
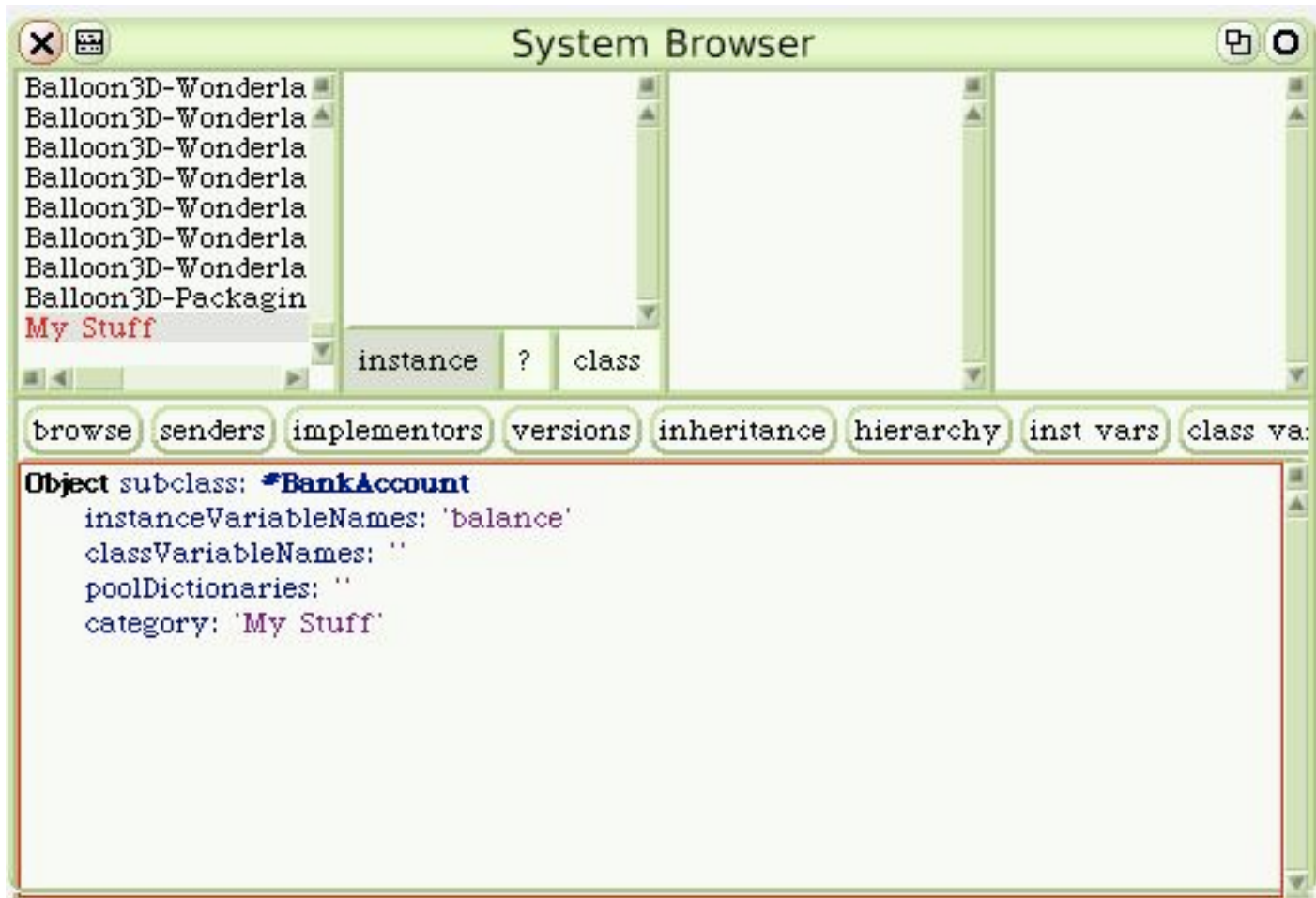
- Developed by Alan Kay et al at Xerox PARC
  - Goal: Build a small, portable computer
    - Remember, computers required separate rooms then

- Key ideas
  - Object oriented programming ideas from Simula
  - Everything is an object
  - Intended for non-experts, especially children
  - Language and language editing environment integrated into operating system

Example

Example (cont.)
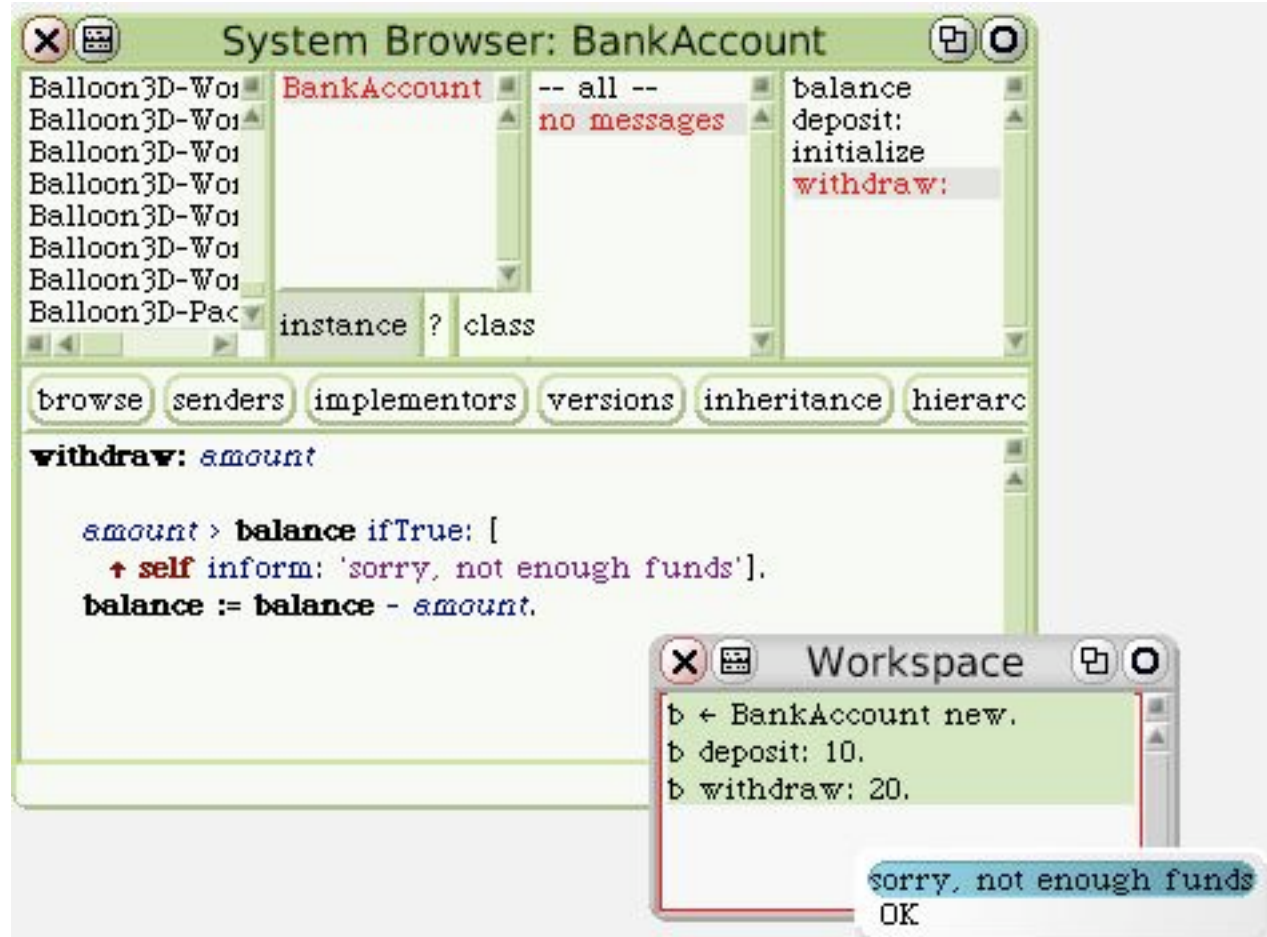
Objective-C (early 1980s)

- Created by Brad Cox & Tom Love
  - Influenced by Smalltalk
  - Popularized by NeXT, acquired by Apple
    - Used in Mac OS X, iOS development

- Imperative, object-oriented language
  - Implemented as C preprocessor
  - Strict superset of C

- Differences from Java
  - Send messages to objects (vs. calling object method)
    - Equivalent to dynamic typing of objects
    - Messages accepted at runtime, may be forwarded to other obj

Objective-C Example / Equiv. Java Code

```
@interface myInt { }
+(int)str2i:(NSString)s;
-(NSString)int2s:(int)i;
@end

@implementation myInt
+(int)str2i:(NSString)s
{
    [o f:s]
}
-(NSString)int2s :(int)i;
{
    NSString *str = @"hello";
    str = [str stringByAppendingString:@" world"];
}
@end
```

```
class myInt {
  static int str2i(String s){
    o.f(s);
  }
  String int2s(int i){
    String str = "hello";
    str = str + " world";
  }
}
```

Parameter type

Method return type

Method invocation

CMSC330 Spring 2020

C++ (1983)

- Bjarne Stroustrup, Bell Labs
- Began as "C with Classes" (~1980)
  - A preprocessor for C code
  - Added Simula-like classes
- Why use C as a base language?
  - C is *flexible* – can write any kind of program
  - C is *efficient* – can efficiently use hardware
  - C is *available* – C compilers exist for most hardware
  - C is *portable* – Porting requires some effort, but doable
    [This and remaining quotes from Stroustrup, *The Design and Evolution of C++*]

C++ Example

```
#include <iostream>
int main ()
{
    int n;
    cout << "Enter number > ";
    cin >> n;
    while (n>0) {
        cout << n << ", ";
        --n;
    }
    cout << "Done!\n";
    return 0;
}
```

Overloaded operators

Stroustrup's Language-Technical Rules

- No implicit violations of the static type system
- Provide as good support for user-defined types as for built-in types
- Locality is good
  - Abstraction and modularity
- Avoid order dependencies
- If in doubt, pick the variant of a feature that is easiest to teach
- Syntax matters (often in perverse ways)
- Preprocessor usage should be eliminated

STL – Standard Template Library (1994)

- C++ library of
  - Container classes, algorithms, iterators
  - Inspired Java class libraries

- A generic library
  - Components are parameterized using templates
  - Java didn't get generics until version 1.5 (2004)

- Example

  ```
  vector<int> v(3);   // vector of 3 int elements
  v[0] = 7;           // overloading [ ] operator
  ```

Very Brief Comparison to Java

- C++ compiled, Java interpreted/JIT'd
  - C++ programs tend to be faster, but gap narrowing
- C++ does not guarantee safety
  - But makes an effort to be better than C
- C++ is much more complicated than Java
  - Copy constructors, assignment operator overloading, destructors, etc.

- Today, C++ still used to build big, commercial applications
  - But Java making inroads because of big library, safety

C# (2001)

- Anders Hejlsberg at Microsoft

- Microsoft's competitor to Java (developed @ Sun)

- Imperative, object-oriented language
  - Syntax based on C++
  - Static typing, strongly typed
  - Hybrid of C++ and Java

- Differences from Java
  - Allows restricted pointers
  - User defined value types (e.g., C structs)
  - SQL-like language integrated queries

C# Example

```
using System;
class Employee { }
class Contractor : Employee { }
class CastExample
{
    public static void Main ()
    {
        Employee e = new Employee();
        Console.WriteLine("e = {0}",
                e == null ? "null" : e.ToString());
        Contractor c  = e as Contractor ;
        Console.WriteLine("c = {0}",
                c == null ? "null" : c.ToString());
    }
}
```

Swift (2014)
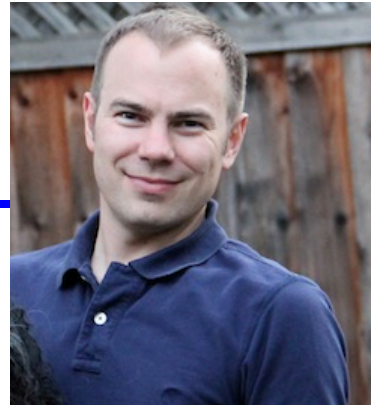
- Chris Lattner at Apple

- Successor to Objective-C

  - Also influenced by Rust, Haskell, Ruby, Python, C#

  - Described as "Objective-C without the C"

  - Code can be mixed with C, C++, Objective-C

- Differences from Objective-C

  - Does not create pointers by default

  - Uses Java-style syntax for calling methods

  - Includes features from Objective-C and other PLs

    - Categories, closures, 1st class functions, type inference, etc.
    - But with cleaner syntax

Swift Example

```
func str2i(s:String) -> int
{
    o->f(i)
}
func int2s(i:int) -> String
{
  var str = "hello"
  str += " world"

  let int: x = 1
  var y = 2
}
```

Method return type

Parameter type

Method invocation

Constant value
Explicit type

Variable value
Type inference

Programming Language Popularity Metrics
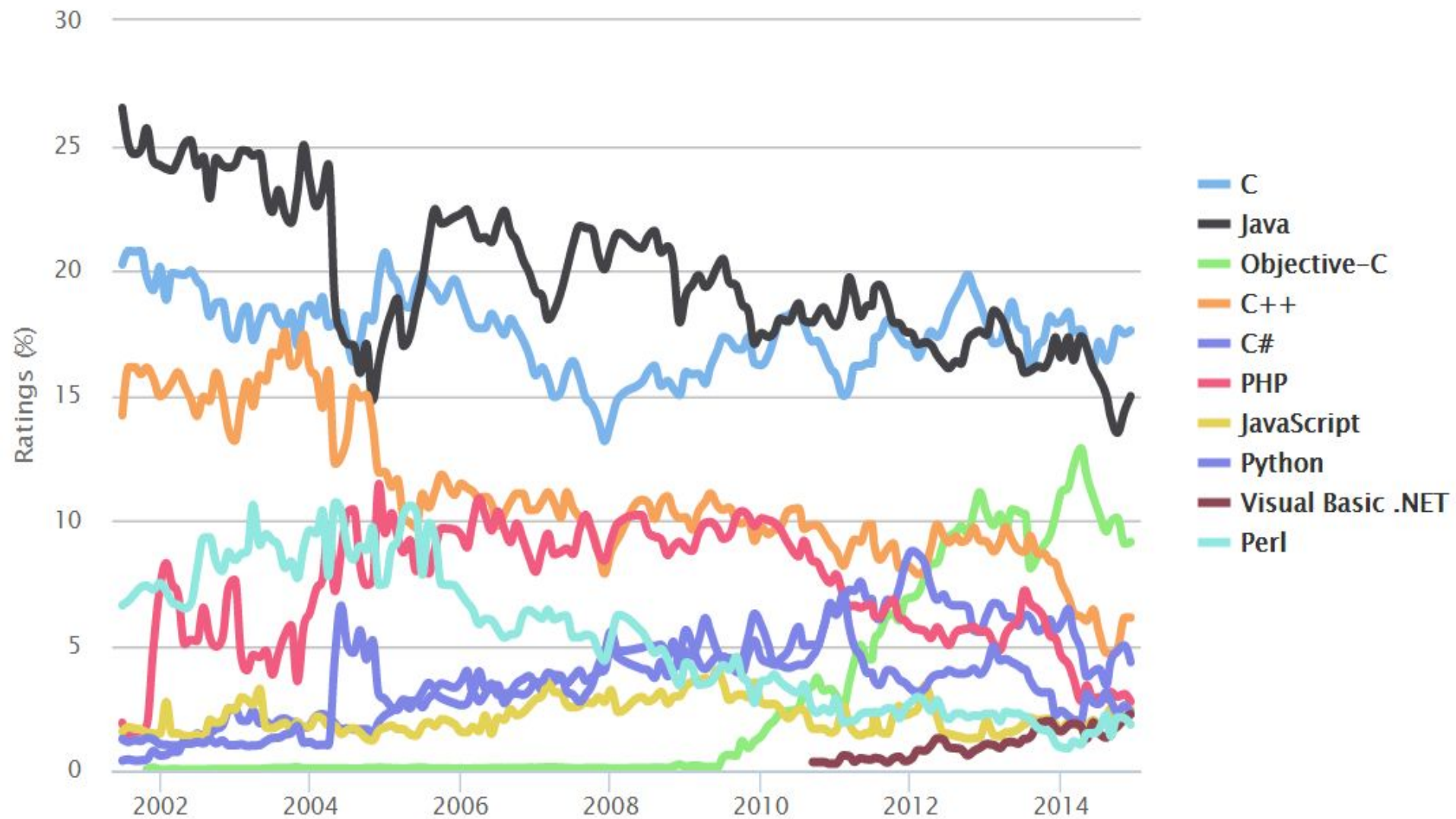
- ## Web references
    - TIOBE Programming Community Index
        (# references in Google, MSN, Yahoo, YouTube)
    - Stack Overflow (Question & answer site for programmers)

- ## Source code
    - SourceForge (Open source projects)
    - GitHub (Hosting service for software development projects)

- ## Computer book sales
    - O'Reilly (publishes programming language books)

TIOBE (Web References)

May 2020	May 2019	Change	Programming Language	Ratings	Change
1	2	⌃	C	17.07%	+2.82%
2	1	⌄	Java	16.28%	+0.28%
3	4	⌃	Python	9.12%	+1.29%
4	3	⌄	C++	6.13%	-1.97%
5	6	⌃	C#	4.29%	+0.30%
6	5	⌄	Visual Basic	4.18%	-1.01%
7	7		JavaScript	2.68%	-0.01%
8	9	⌃	PHP	2.49%	-0.00%
9	8	⌄	SQL	2.09%	-0.47%
10	21	⌃⌃	R	1.85%	+0.90%
11	18	⌃⌃	Swift	1.79%	+0.64%
12	19	⌃⌃	Go	1.27%	+0.15%
13	14	⌃	MATLAB	1.17%	-0.20%
14	10	⌄⌄	Assembly language	1.12%	-0.69%
15	15		Ruby	1.02%	-0.32%
16	20	⌃⌃	PL/SQL	0.99%	-0.03%
17	16	⌄	Classic Visual Basic	0.89%	-0.43%
18	13	⌄⌄	Perl	0.88%	-0.51%
19	28	⌃⌃	Scratch	0.83%	+0.32%
20	11	⌄⌄	Objective-C	0.80%	-0.83%

TIOBE Programming Community Index

Source: www.tiobe.com

Legend:
- C
- Java
- Objective-C
- C++
- C#
- PHP
- JavaScript
- Python
- Visual Basic .NET
- Perl

CMSC330 Spring 2020

TIOBE – Long Term Popularity

Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
Java	1	2	1	2	3	-	-	-
C	2	1	2	1	1	2	1	1
Python	3	7	6	7	23	22	-	-
C++	4	4	4	3	2	1	3	9
C#	5	5	5	8	8	-	-	-
JavaScript	6	8	8	9	5	-	-	-
PHP	7	6	3	4	25	-	-	-
SQL	8	-	-	-	-	-	-	-
Swift	9	95	-	-	-	-	-	-
Ruby	10	11	9	23	31	-	-	-
Objective-C	12	3	14	37	-	-	-	-
Lisp	28	21	15	13	7	5	4	2
Fortran	29	27	21	14	16	4	2	12
Ada	34	26	24	16	15	6	5	3
Pascal	238	15	12	46	12	3	7	5

Interesting Trends

- C, C++, Java most popular
    - For last 15 years
- Objective C way up
    - Wasn't even in the top 20 in 2008
    - Used for programming Apple iOS devices
- Four scripting languages in top 10
    - PHP, Python, JavaScript, Perl
- F# is only functional language in top 20
    - Multi-paradigm functional language w/ type inference

Summary

- Many programming languages
- Wide variety of language features
  - Evaluated through trial and error
- Languages you should now know

Language	Paradigm	Type System	Type Checking	Variable Declarations	Type Declarations
Java	OO/Imp	Strong	Static	Explicit	Explicit
C	Imperative	Weak	Static	Explicit	Explicit
Ruby	OO/Scripting	Strong	Dynamic	Implicit	None
OCaml	Functional	Strong	Static	Explicit	Inferred
Rust	Imp/Func	Strong	Static	Explicit	Expl/Infer

GitHub (Repositories)

- Top repositories created as of Oct 2013

Rank	Language	#	Rank	Language	#
1	JavaScript	264K	11	CSS	18K
2	Ruby	219K	12	Perl	15K
3	Java	158K	13	CoffeeScript	11K
4	PHP	114K	14	VimL	8K
5	Python	95K	15	Scala	7K
6	C++	78K	16	Go	7K
7	C	68K	17	Prolog	6K
8	Objective-C	36K	18	Clojure	5K
9	C#	32K	19	Haskell	5K
10	Shell	28K	20	Lua	4K

GitHub (New Repositories 2008-2013)

New GitHub repositories

RedMonk

Percentage (annual)

20

15

10

5

0

JavaScript
Ruby
Java
PHP
Python
C/C++
C#/Obj-C
Perl

2008 2009 2010 2011 2012 2013
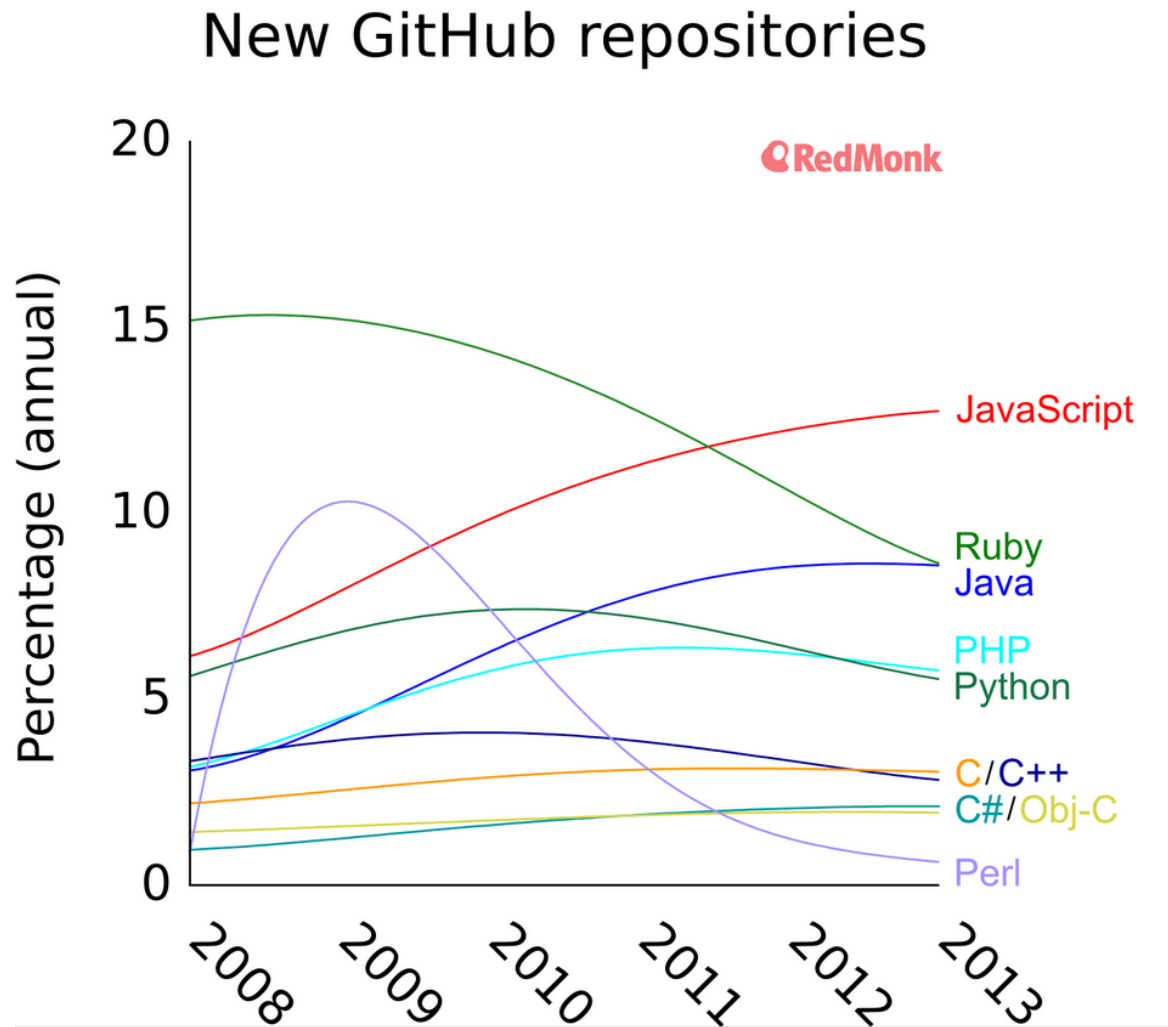
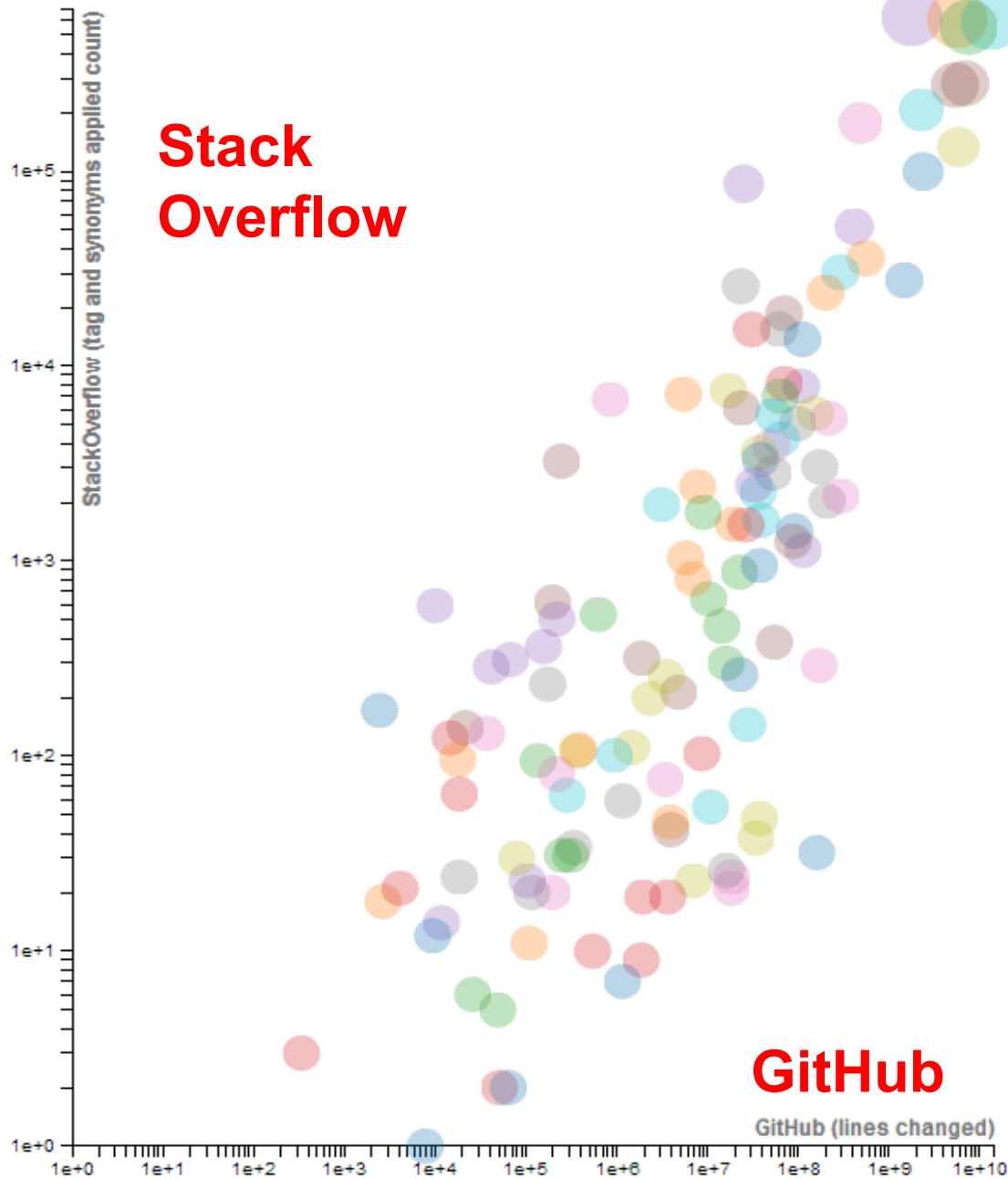# GitHub (New Users, Issues, Repositories)

New users, issues, and repos
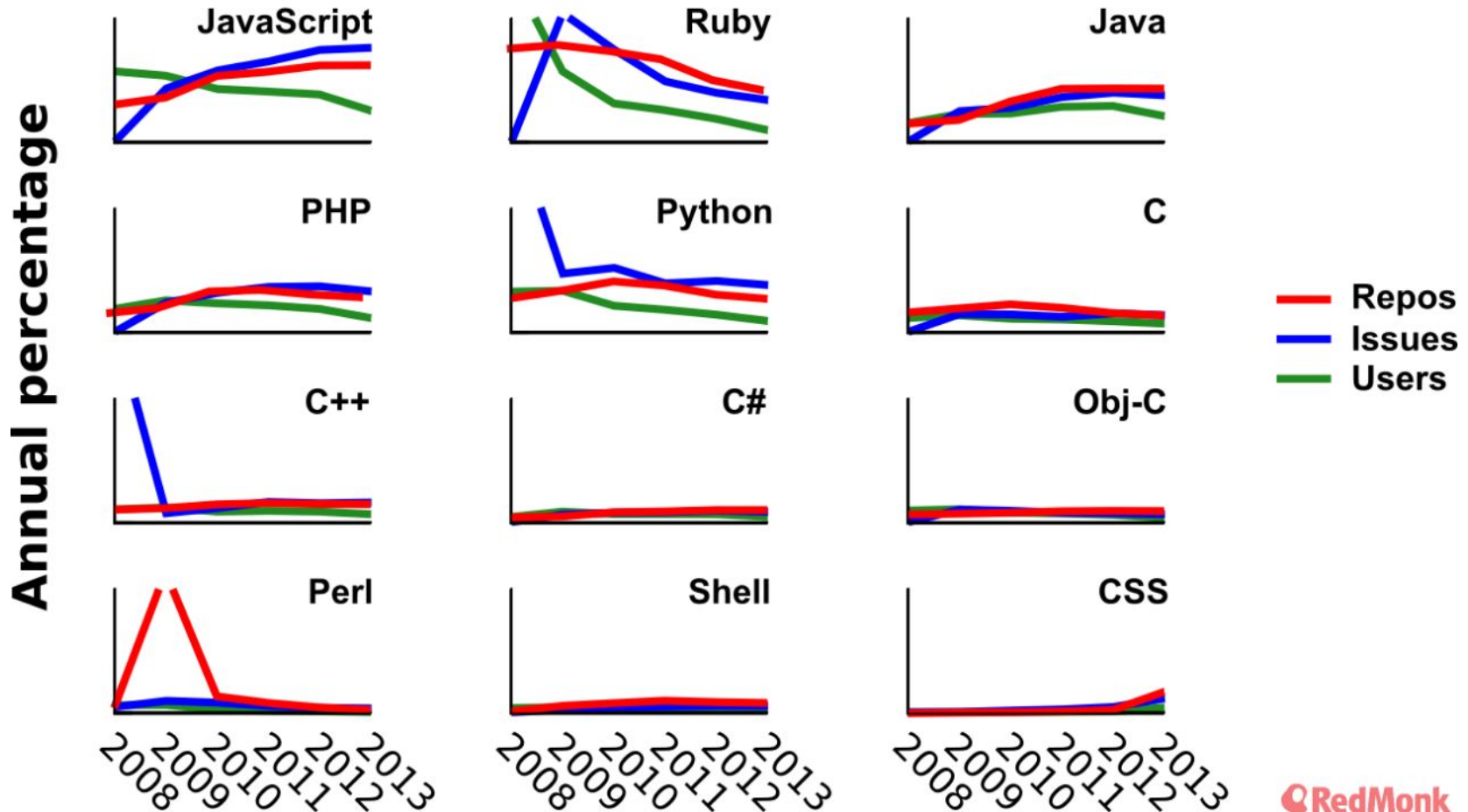
Combining Code & Web References

- From langpop.corger.nl

- Combined metric based on
    - GitHub
        - Lines of code modified
    - StackOverflow
        - # tags & references in posts

StackOverflow (tag and synonyms applied count)

Stack Overflow

GitHub

GitHub (lines changed)

C#	Java	JavaScript
PHP	Python	C++
CSS	Objective-C	C
Ruby	XML	R
Perl	Matlab	Shell
Delphi	Scala	XSLT
Haskell	PowerShell	Assembly
ActionScript	Groovy	ColdFusion
Awk	Clojure	Io
F#	Lua	CoffeeScript
Visual Basic	Go	Erlang
Prolog	Arduino	AppleScript
Scheme	FORTRAN	Dart
OCaml	Tcl	Max
TypeScript	Emacs Lisp	ANTLR
VHDL	Racket	Processing
Common Lisp	XQuery	Verilog
D	AutoHotkey	Puppet
AutoIt	Pascal	Ada
Smalltalk	Vala	TeX
COBOL	RobotFramew	Self
Apex	Rebol	IDL
ABAP	Coq	Factor
DOT	Rust	NetLogo

CMSC330 Spring 2020

GitHub + StackOverflow

- Top 15 languages as of March 2014

Rank	Language	Ratings	Rank	Language	Ratings
1	C#	15.39%	11	XML	2.15%
2	Java	15.05%	12	R	1.3%
3	Javascript	14.56%	13	Perl	0.89%
4	PHP	13.7%	14	Matlab	0.76%
5	Python	7.06%	15	Shell	0.69%
6	C++	6.91%		…	
7	CSS	5.16%	28	F#	0.15%
8	Objective-C	4.42%	31	Visual Basic	0.14%
9	C	3.33%	34	Prolog	0.10%
10	Ruby	2.49%	40	OCaml	0.06%

O'Reilly (Computer Book Sales)

O'Reilly (Book Sales 2011 vs. 2010)

Showing: Programming Language Quarter/Previous Year Units