Welcome to Programming Languages

CSCI 3136: Principles of Programming Languages

Agenda

- Announcements
  - Get a Top Hat account
  - Check your Dal email
  - Check your Brightspace account
- Lecture Contents
  - Administrivia
  - Introduction to Programming Languages

Course Description

- Overview
- Lexical Analysis
- Parsing
- Semantic Analysis
- Naming and Binding
- Flow Control
- Computation Abstraction
- Type Systems and Memory Management
- Functional Languages

Administrivia

- Who/Where/When
  - Instructor: Alex Brodsky (person at front of room)
  - Location: MACME (right here)
  - Meet Time: WF 10:05 – 11:25 (now)
- Pre-reqs: CSCI 2110, CSCI 2112, CSCI 2132
- Contacts:
  - Email: csci3136@dal.ca
  - LMS (website): http://dal.brightspace.com
  - Office hours: Initially:
    - WF 14:30 – 15:30
- Optional Text: Scott M., "Programming Languages Pragmatics, 4th ed.", Morgan Kaufmann, 2015, ISBN: 0124104096.
  - Earlier editions are fine.

Class Rules

- Cell phones and beepers should be SILENT.
- I will start class at 10:05 and finish by 11:25.
- Coming late is fine as long as you do not disturb the class.
- I am hard of hearing, so I may ask you to repeat yourself.
- If my writing becomes too messy let me know.

Course Assessment

- Assessment Components:
    - Participation and Quizzes (5%) via TopHat
    - Best 8 of 10 assignments: 20% (2.5% each)
    - Optional midterm: (25% or 0%)
    - Final exam: (50% or 75%)

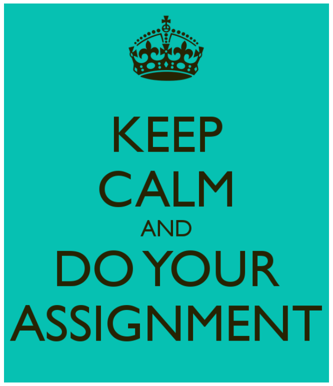
  I will choose the marking scheme that benefits each student.

- Midterm: Wednesday, June 19, 10:05 – 11:25, in CHEB 170.
- Final Exam: Scheduled during week of July 31 - August 6.

Top Hat Is Required

- We will be using the Top Hat Student Response System in class and in tutorial
- You will need to:
    - Register for Top Hat at http://www.tophat.com
        - If you are not already registered
    - Join the course. Join code: 925103
- Why do we use Top Hat?
    - Facilitates ongoing small-stakes assessment and feedback
    - Fairly assesses attendance and participation

Assignments



- Best 8 out of 10 assignments
- Due at 9am on
  - May 24 and 31
  - June 7, 14, and 28
  - July 5, 12, 19, 26
- **No late submissions accepted.**
- All assignments must be submitted via Brightspace
  - Written assignments should be scanned and a PDF submitted.
- Assignments may be done in groups of up to three students.

What is Expected of Me?

- I don’t know/remember how to do proofs!
  - You are responsible for knowing/relearning/learning what was covered in CSCI 2112.
- I started the assignment yesterday!
  - You need to start working on the assignments early.
- I spent 20 hours on this assignment and could not figure out what to do or where to start!
  - If you are spinning your wheels, please come and talk to me.
- I spent the last 10 hours debugging this assignment!
  - See above.
- I will use the posted power point slides and not take notes
  - These slides are incomplete. Please take notes during lectures

What Should I Know?

- Programming
- Mathematical proofs
- Testing and Debugging

Self-Assessment Quiz

- There is a self-assessment quiz on Brightspace.
- Used to gauge the expectations in this course.
- Do this on your own and compare to the posted solution.

What Do I Need to Do Now?

- Self-Assessment quiz [Recommended]
- Get a Top Hat account. [Required]
  - We will start using Top Hat on **Friday**.
- Check Brightspace regularly. [Required]
- **Check my Dal Email. [Required]**

Course Representative

- Our Course Rep is **???**: ???@dal.ca
- The Course Representative is a point of contact to facilitate and provide more timely feedback mechanisms to instructors and to the Faculty of Computer Science.
- Additionally, Course Representatives can assist peers in navigating to the most appropriate support mechanism on campus. You can think of a CR as ‘the middle person’; a neutral point of contact for students to use when they don’t feel comfortable addressing an issue with the professor directly.

I really hate this slide.

30%

10%

15%

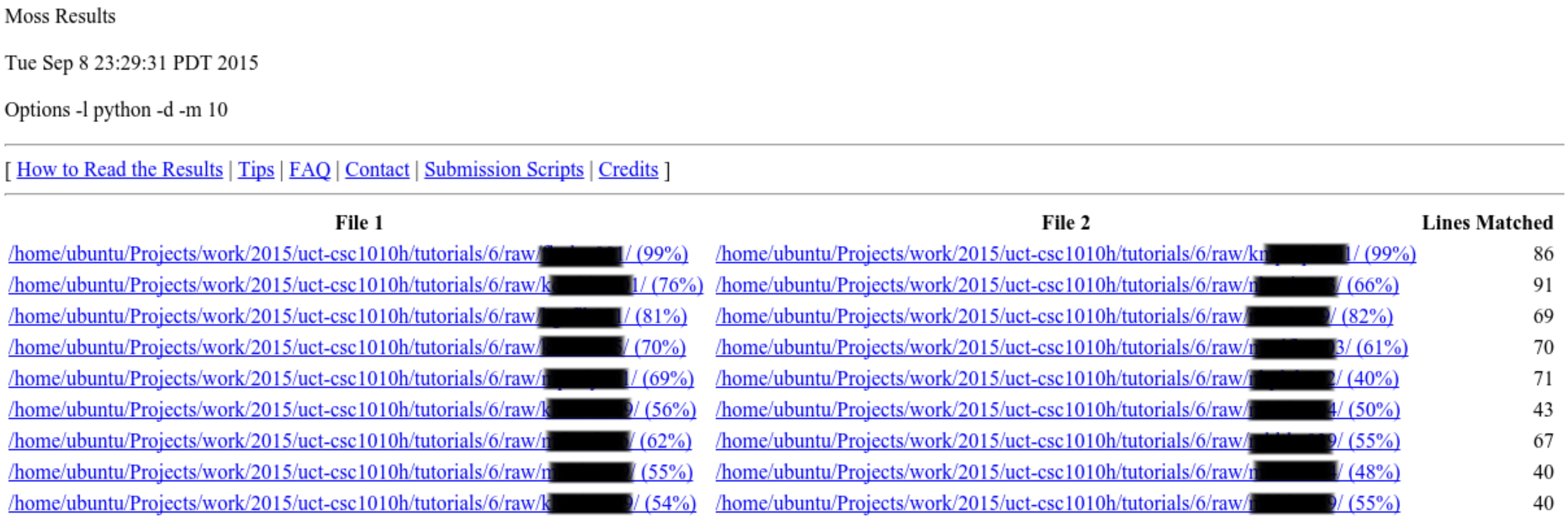
13%

Academic Integrity

- Academic integrity means being honest in the fulfillment of your academic responsibilities thus establishing mutual trust.
- Violations of intellectual honesty are offensive to the entire academic community, not just to the individual faculty member and students in whose class an offense occurs.
  - E.g., cheating on tests, plagiarism, falsification of experimental data, etc.
- All cases of academic misconduct are automatically referred to the Faculty Academic Integrity Officer.
- Suggested Guidelines:
  - Put pencils and pens away when discussing problem with other people
  - Acknowledge any help you received in your assignments: state name of person.
  - Write your own code! You may look at code all you want, but don’t cut and paste!

Moss: Software Similarity Detection Software
https://theory.stanford.edu/~aiken/moss/

All submitted code will be passed through Moss which performs a pair-wise comparison of similarities



Moss Results

Tue Sep 8 23:29:31 PDT 2015

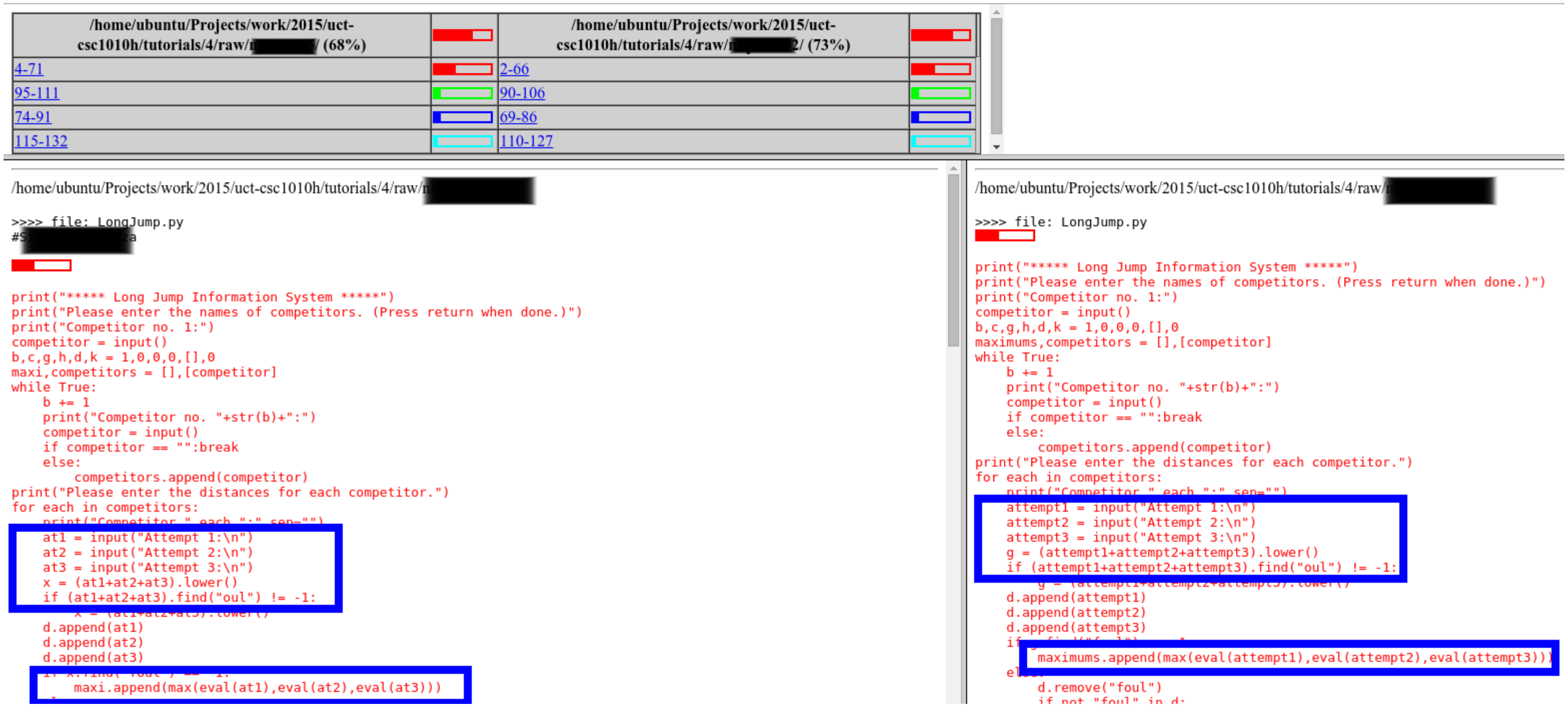
Options -l python -d -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (99%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (99%)	86
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (76%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (66%)	91
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (81%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (82%)	69
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (70%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (61%)	70
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (69%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (40%)	71
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (56%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (50%)	43
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (62%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (55%)	67
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (55%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (48%)	40
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (54%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (55%)	40

Moss: Software Similarity Detection Software
https://theory.stanford.edu/~aiken/moss/

Moss even identifies which parts are similar.



/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]/ (68%)	🟥	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]2/ (73%)	🟥
4-71	🟥	2-66	🟥
95-111	🟩	90-106	🟩
74-91	🟦	69-86	🟦
115-132	▭	110-127	▭

/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]

```
>>>> file: LongJump.py
#[redacted]

print("***** Long Jump Information System *****")
print("Please enter the names of competitors. (Press return when done.)")
print("Competitor no. 1:")
competitor = input()
b,c,g,h,d,k = 1,0,0,0,[],0
maxi,competitors = [],[competitor]
while True:
    b += 1
    print("Competitor no. "+str(b)+":")
    competitor = input()
    if competitor == "":break
    else:
        competitors.append(competitor)
print("Please enter the distances for each competitor.")
for each in competitors:
    print("Competitor ", each ":" sep="")
    at1 = input("Attempt 1:\n")
    at2 = input("Attempt 2:\n")
    at3 = input("Attempt 3:\n")
    x = (at1+at2+at3).lower()
    if (at1+at2+at3).find("oul") != -1:
        x = (at1+at2+at3).lower()
    d.append(at1)
    d.append(at2)
    d.append(at3)
    if x.find("foul") == -1:
        maxi.append(max(eval(at1),eval(at2),eval(at3)))
```

/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]

```
>>>> file: LongJump.py

print("***** Long Jump Information System *****")
print("Please enter the names of competitors. (Press return when done.)")
print("Competitor no. 1:")
competitor = input()
b,c,g,h,d,k = 1,0,0,0,[],0
maximums,competitors = [],[competitor]
while True:
    b += 1
    print("Competitor no. "+str(b)+":")
    competitor = input()
    if competitor == "":break
    else:
        competitors.append(competitor)
print("Please enter the distances for each competitor.")
for each in competitors:
    print("Competitor ", each ":" sep="")
    attempt1 = input("Attempt 1:\n")
    attempt2 = input("Attempt 2:\n")
    attempt3 = input("Attempt 3:\n")
    g = (attempt1+attempt2+attempt3).lower()
    if (attempt1+attempt2+attempt3).find("oul") != -1:
        g = (attempt1+attempt2+attempt3).lower()
    d.append(attempt1)
    d.append(attempt2)
    d.append(attempt3)
    if g.find("foul") == -1:
        maximums.append(max(eval(attempt1),eval(attempt2),eval(attempt3)))
    else:
        d.remove("foul")
        if not "foul" in d:
```

If a student does not wish their assignments to be submitted to Moss, they should contact the instructor.

Never …

- Never email/send/copy your work to another student before the due date
- Never **use homework sites such as Chegg or Course Hero**
  - If you are using the site, chances are another student is also …
- Never copy and paste someone else’s code into your own assignment
- Never type code from another source into your own assignment
- Never write code while discussing the problem with other students

It’s OK to …

- Discuss an assignment with another student or TA
- Ask for help to debug your code (but don’t expect someone else to do the work)
- Explain to someone else how to solve a problem
- Use a whiteboard/blackboard/scrap paper to work out the problem, as long as you erase/destroy the board/paper after the discussion
- Look at other people’s code, BUT take a brief break before writing your own

Typical Penalties for First-Time Offences ...

It is always better to take a 0 then to be caught.

- Plagiarism:
  - 0 on the assignment
  - 6-month notation on transcript
  - Minor grade decrement, e.g. B→B-, C→C-, D→F
- Contributing to the Commission of Plagiarism
  - 6-month notation on transcript
  - Minor grade decrement, e.g. B→B-, C→C-, D→F
- Cheating on a test / practicum
  - 0 on the test
  - 12-month notation on transcript
  - Minor grade decrement, e.g. B→B-, C→C-, D→F

Chances of being caught?

High!

Culture of Respect

- Every person has a right to respect and safety.
- Inclusiveness is fundamental to education and learning.
- Misogyny and other disrespectful behaviour in our classrooms, on our campus, on social media, and in our community is unacceptable.
- What to do:
  - Be Ready to Act
  - Identify the Behaviour
  - Appeal to Principles
  - Set Limits
  - Find or be an Ally
  - Be Vigilant

Why are we here?

Motivation

Programmer's Dream

Natural language instructions:

"Computer schedule my courses for next year."

- Expressive
- Easy
- Ambiguous

Computer Reality

Computers take very primitive instructions:

```
add $42, %ebp
movl %eax, %ecx
```

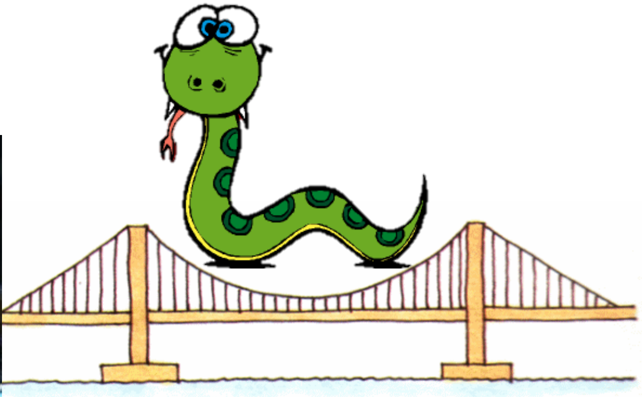
- Simple and specific
- Hard and Tedious
- Unambiguous

Programmer Reality

- Programming Languages bridge the divide

```
for i in range(1, 10)
  print i
```

  - Unambiguous
  - Expressive
  - Less Tedious



What’s this Course About?

- Program Translation:
  - Computers only understand the low level
  - Need to translate all programs into the computer’s language
- Language Features:
  - What features should a useful language have?
  - What tasks is a language suited for?
  - How do we implement these features?

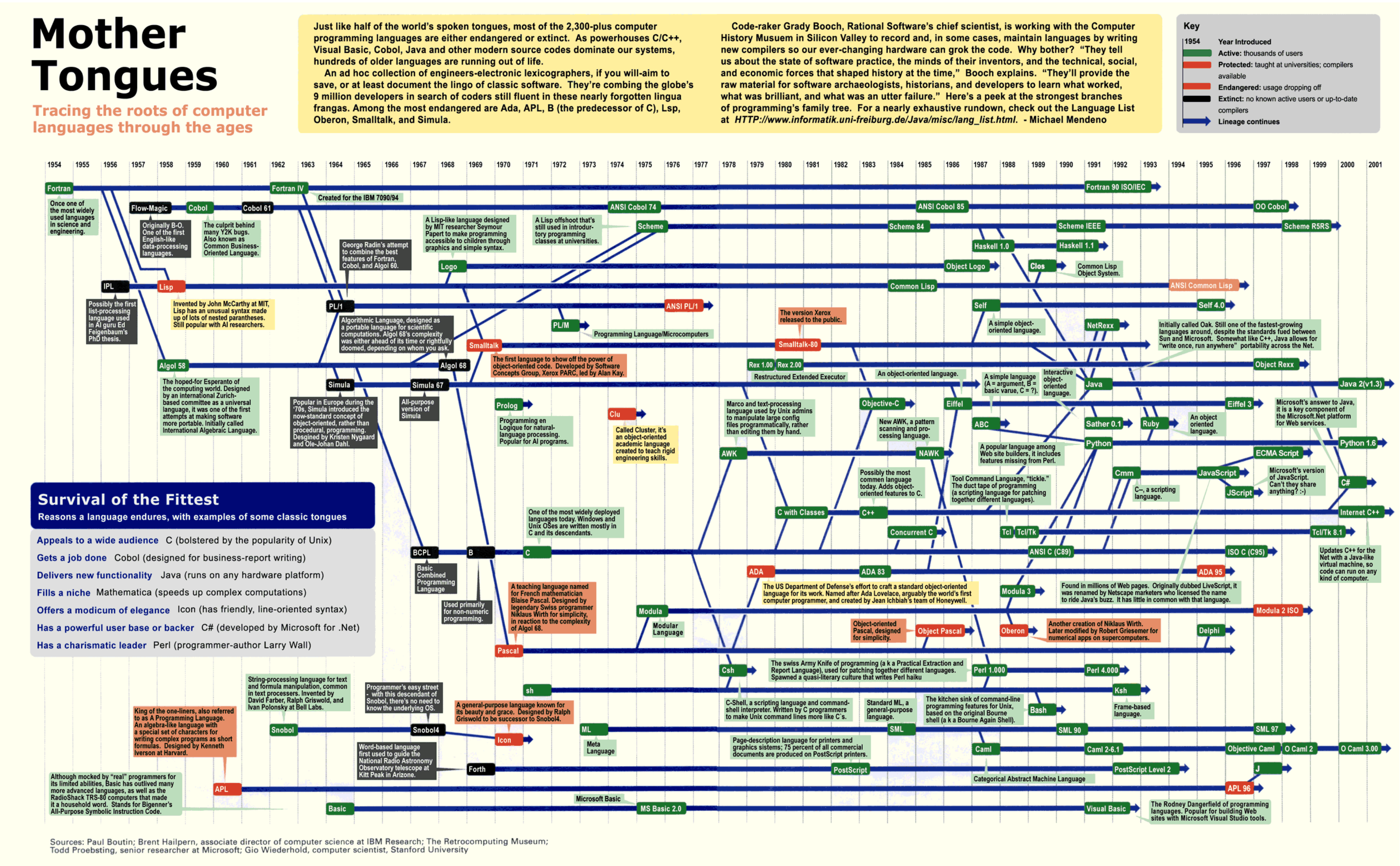
Why Do We Care?

- Idea: Languages are like tools in a toolbox.
    - Different languages for different tasks.
    - Sometimes we need to build our own for the task at hand.
- Also …
    - Easier to learn new languages
    - Make better choices when deciding which language to use
    - Simulate useful features in other languages
    - Use languages more effectively

A Brief History of Languages

Stone Age	Machine language → assembly language
Scientific	FORTRAN (1957) → versions IV, . . . 1995
Structural	ALGOL (1958) → Pascal (1971) Ada (1983)
Functional	Lisp (1959) → Scheme (1975), Common Lisp (1984)
Business	COBOL (1959), APL (1960), SNOBOL (1962)
Home/Hobbiest	BASIC (1964) → Visual Basic (1990)
Object Oriented	Simula (1967) → Smalltalk (1980) → C++ (1985)
Imperative	C (1972) → C++ (1985), Java (1995)
Scripting	Perl (1987), Python (1990s), JavaScript (1995), PHP (1995), C# (2002)
Declarative	Prolog (1973)

If you can read this you don’t need glasses



Mother Tongues

Tracing the roots of computer languages through the ages

Just like half of the world’s spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They’re combing the globe’s 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software’s chief scientist, is working with the Computer History Musuem in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? “They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time,” Booch explains. “They’ll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure.” Here’s a peek at the strongest branches of programming’s family tree. For a nearly exhaustive rundown, check out the Language List at *HTTP://www.informatik.uni-freiburg.de/Java/misc/lang_list.html.* - Michael Mendeno

Key

1954 **Year Introduced**

🟩 **Active:** thousands of users

🟥 **Protected:** taught at universities; compilers available

🟥 **Endangered:** usage dropping off

⬛ **Extinct:** no known active users or up-to-date compilers

➡ **Lineage continues**

Once one of the most widely used languages in science and engineering.

Created for the IBM 7090/94

Originally B-O. One of the first English-like data-processing languages.

The culprit behind many Y2K bugs. Also known as Common Business-Oriented Language.

George Radin’s attempt to combine the best features of Fortran, Cobol, and Algol 60.

A Lisp-like language designed by MIT researcher Seymour Papert to make programming accessible to children through graphics and simple syntax.

A Lisp offshoot that’s still used in introdur-tory programming classes at universities.

Common Lisp Object System.

Possibly the first list-processing language used in AI guru Ed Feigenbaum’s PhD thesis.

Invented by John McCarthy at MIT, Lisp has an unusual syntax made up of lots of nested parantheses. Still popular with AI researchers.

The version Xerox released to the public.

Algorithmic Language, designed as a portable language for scientific computations. Algol 68’s complexity was either ahead of its time or rightfully doomed, depending on whom you ask.

Programming Language/Microcomputers

A simple object-oriented language.

Initially called Oak. Still one of the fastest-growing languages around, despite the standards fued between Sun and Microsoft. Somewhat like C++, Java allows for “write once, run anywhere” portability across the Net.

The first language to show off the power of object-oriented code. Developed by Software Concepts Group, Xerox PARC, led by Alan Kay.

Restructured Extended Executor

An object-oriented language.

A simple language (A = argument, B = basic varue, C = ?).

Interactive object-oriented language.

The hoped-for Esperanto of the computing world. Designed by an international Zurich-based committee as a universal language, it was one of the first attempts at making software more portable. Initially called International Algebraic Language.

Popular in Europe during the ‘70s, Simula introduced the now-standard concept of object-oriented, rather than procedural, programming. Desgined by Kristen Nygaard and Ole-Johan Dahl.

All-purpose version of Simula

Programming en Logique for natural-language processing. Popular for AI programs.

Called Cluster, it’s an object-oriented academic language created to teach rigid engineering skills.

Marco and text-processing language used by Unix admins to manipulate large config files programmatically, rather than editing them by hand.

New AWK, a pattern scanning and pro-cessing language.

An object oriented language.

Microsoft’s answer to Java, it is a key component of the Microsoft.Net platform for Web services.

A popular language among Web site builders, it includes features missing from Perl.

Possibly the most commen language today. Adds object-oriented features to C.

Tool Command Language, “tickle.” The duct tape of programming (a scripting language for patching together different languages).

C--, a scripting language.

Microsoft’s version of JavaScript. Can’t they share anything? :-)

One of the most widely deployed languages today. Windows and Unix OSes are written mostly in C and its descendants.

Updates C++ for the Net with a Java-like virtual machine, so code can run on any kind of computer.

Basic Combined Programming Language

Used primarily for non-numeric programming.

The US Department of Defense’s effort to craft a standard object-oriented language for its work. Named after Ada Lovelace, arguably the world’s first computer programmer, and created by Jean Ichbiah’s team of Honeywell.

Found in millions of Web pages. Originally dubbed LiveScript, it was renamed by Netscape marketers who licensed the name to ride Java’s buzz. It has little in common with that language.

A teaching language named for French mathematician Blaise Pascal. Designed by legendary Swiss programmer Niklaus Wirth for simplicity, in reaction to the complexity of Algol 68.

Modular Language

Object-oriented Pascal, designed for simplicity.

Another creation of Niklaus Wirth. Later modified by Robert Griesemer for numerical apps on supercomputers.

The swiss Army Knife of programming (a k a Practical Extraction and Report Language), used for patching together different languages. Spawned a quasi-literary culture that writes Perl haiku

String-processing language for text and formula manipulation, common in text processers. Invented by David Farber, Ralph Griswold, and Ivan Polonsky at Bell Labs.

Programmer’s easy street - with this descendant of Snobol, there’s no need to know the underlying OS.

A general-purpose language known for its beauty and grace. Designed by Ralph Griswold to be successor to Snobol4.

C-Shell, a scripting language and command-shell interpreter. Written by C programmers to make Unix command lines more like C`s.

Standard ML, a general-purpose language.

The kitchen sink of command-line programming features for Unix, based on the original Bourne shell (a k a Bourne Again Shell).

Frame-based language.

Meta Language

King of the one-liners, also referred to as A Programming Language. An algebra-like language with a special set of characters for writing complex programs as short formulas. Designed by Kenneth Iverson at Harvard.

Word-based language first used to guide the National Radio Astronomy Observatory telescope at Kitt Peak in Arizone.

Page-description language for printers and graphics sistems; 75 percent of all commercial documents are produced on PostScript printers.

Categorical Abstract Machine Language

Although mocked by “real” programmers for its limited abilities, Basic has outlived many more advanced languages, as well as the RadioShack TRS-80 computers that made it a household word. Stands for Bigenner’s All-Purpose Symbolic Instruction Code.

Microsoft Basic

The Rodney Dangerfield of programming languages. Popular for building Web sites with Microsoft Visual Studio tools.

Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

Appeals to a wide audience	C (bolstered by the popularity of Unix)
Gets a job done	Cobol (designed for business-report writing)
Delivers new functionality	Java (runs on any hardware platform)
Fills a niche	Mathematica (speeds up complex computations)
Offers a modicum of elegance	Icon (has friendly, line-oriented syntax)
Has a powerful user base or backer	C# (developed by Microsoft for .Net)
Has a charismatic leader	Perl (programmer-author Larry Wall)

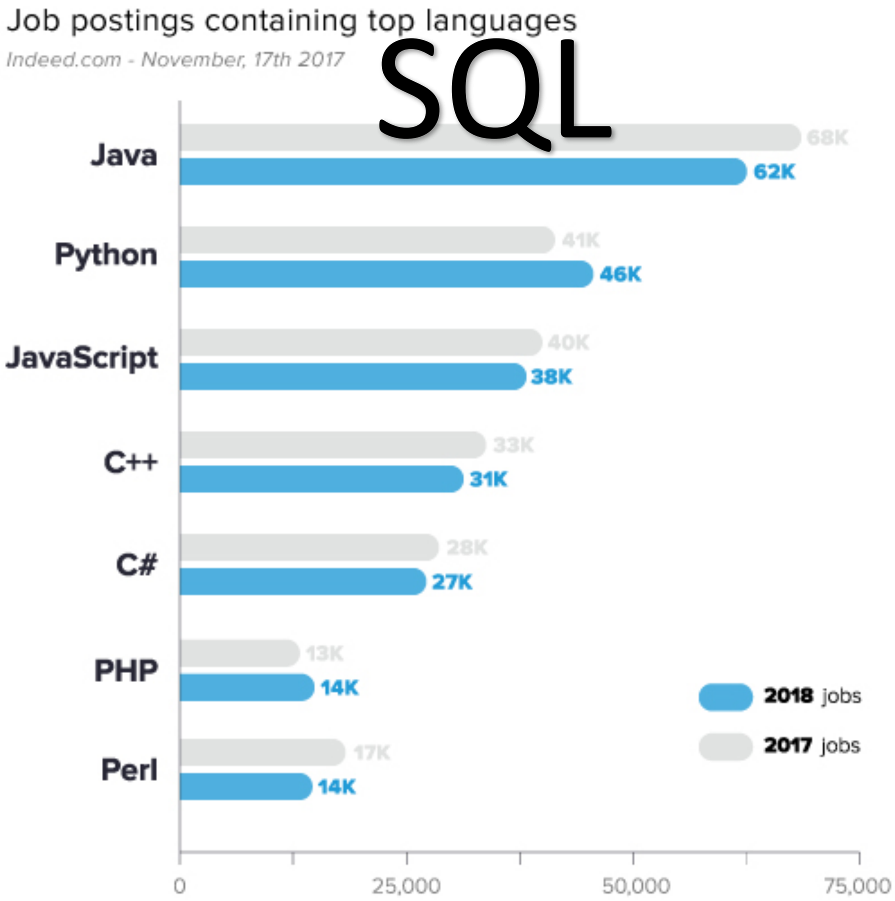
Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

Why so many languages?

- Evolution
- Specific purposes
- Programmer preference

What make languages popular?

- Availability
- Learnability
- Expressiveness
- Ease of use
- Install base



Job postings containing top languages

Indeed.com - November, 17th 2017

SQL

https://www.codingdojo.com/blog/7-most-in-demand-programming-languages-of-2018/

How do we categorize languages?

- Answer: Programming Paradigms
  - **Imperative** : (von Neunmann model)
    - FORTRAN
    - COBOL
    - BASIC
    - OO languages
  - **Functional** :
    - Lisp,
    - Scheme,
    - ML
  - **Declarative** :
    - Prolog
    - Visicalc
    - Spreadsheets