

Course Introduction

Programming Languages

CS 214



Welcome!

Welcome to CS 214:

Programming Language Concepts

On-line Course Materials and Syllabus:

<https://cs.calvin.edu/courses/cs/214/meyer/>



Programming Language (PL) History

PL history can be divided into *generations*:

- In the first generation, programming consisted of writing programs in *binary machine language*

```
0010 1011 1000 0000
0010 1111 1000 0100
0011 0011 1000 1000
```

Such programs were

- difficult to write
- prone to programmer errors
- difficult to debug

so people quickly sought a better way...



The Second Generation

To avoid having to program using binary operations, programmers developed *mnemonics* for the operations, and used *symbolic names* instead of memory addresses:

MOV	→	0010	1011	I	→	1000	0000
ADD	→	0010	1111	J	→	1000	0100
STO	→	0011	0011	K	→	1000	1000

MOV	I	→	0010	1011	1000	0000
ADD	J	→	0010	1111	1000	0100
STO	K	→	0011	0011	1000	1000

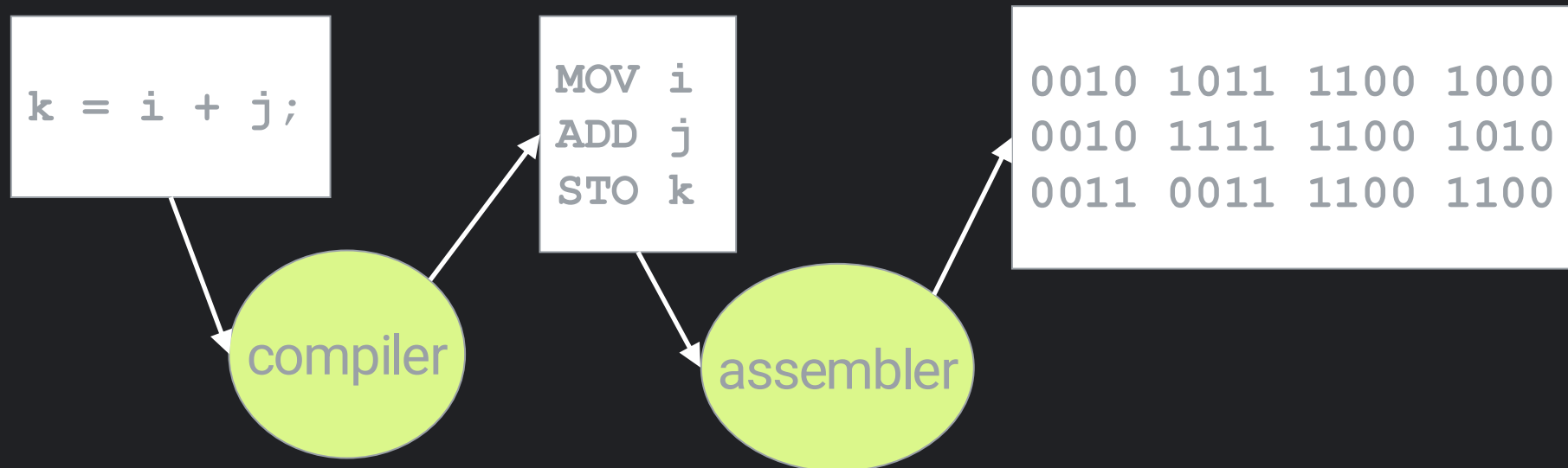
The resulting *assembly languages* were the 2nd gen. PLs.

Assemblers are programs that translate assembly to binary.



The Third Generation

Assembly languages were still awkward and not portable, so Grace Hopper proposed *high level languages* (HLLs) that would hide/be independent of the machine-level details; plus a *compiler* to translate them into machine language:



HLLs are the 3rd generation of programming languages.



HLLs (1940s)

Between 1942 and 1945, Konrad Zuse designed *Plankalkül* for his Z-series of computers in Germany:

- Plan + kalkül = calculus (formal system) for planning
- The very first HLL, it included many advanced features:
 - Assignment statements (local variables only)
 - Conditional statements (and expressions)
 - Iteration (for loops, while loops)
 - Subroutines (non-recursive) and parameters (pass-by-value only)
 - Assertions
 - Exceptions
 - Data structures (arrays, tuples/records, graphs)
 - ...
- Largely unknown in the west until 1970s, due to WW-II...



HLLs (50s)

In the west, there were no HLLs until the late 1950s:

- 1957: John Backus et al design *FORTRAN*
 - FORmula TRANslation; for scientific users
 - Subsequent versions in '58, '62, '77, '90, '95
- 1959: Grace Hopper et al design *COBOL*
 - Emphasized readability; for business users
 - Introduced If-Then-Else statement
- 1959: John McCarthy designs *Lisp*
 - LISt Processing; introduced the linked *list* as primitive type
 - First *functional* programming language (every op is a function)
 - Also introduced dynamic scope, garbage collection



HLLs (60s)

- 1960: Nicholas Wirth et al design *Algol-60*
 - ALGOrithmic Language: designed for encoding algorithms
 - Introduces *block structure*: basis for structured programming
 - Introduces *procedures*: basis for procedural programming
- 1964: Wirth designs *Algol-W* (CASE statement)
 - Kemeny and Kurtz design *BASIC*
 - Iverson designs *APL* (*multidimensional arrays, graphic symbols*)
- 1965: IBM designs *PL-1* (*multi-tasking, exceptions*)
- 1967: Nyquist & Dahl design *Simula*
 - SIMULAtion HLL; introduces *class* construct
 - Laid foundation for O-O programming



Edsger Dijkstra Quotes

- “The use of **COBOL** cripples the mind; its teaching should, therefore, be regarded as a criminal offense.”
- “It is practically impossible to teach good programming to students who have had prior exposure to **BASIC**, as potential programmers they are mentally mutilated beyond hope of regeneration.”
- “**APL** is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.”



HLLs (late 60s, early 70s)

- 1968: Wirth designs *Algol-68*
 - Introduces *If-Then-Elseif* statement; *pointer variables*
- 1970: Wirth designs *Pascal*
 - Consolidates Algol-x features into one simple language
- 1970: Ritchie designs *C* for OS implementation (Unix)
 - Provides direct hardware access, separate compilation/linking,...
 - AKA “high level assembly language” and “Fortran done right”
- 1972: Colmerauer, Roussel & Kowalski design *Prolog*
 - PROgramming LOGic; designed for logic programming
 - Logic/predicate-based HLL for programming *inferences*
 - First logic HLL; used in expert systems
- 1973: SQL by IBM



HLLs (late 70s)

- 1977: Gordon & Milner design *ML* (MetaLanguage)
 - Hybrid HLL: functional, with inference rules (Prolog)
 - Allows linked ADTs to be defined without pointers (recursively)
 - AKA a “higher level language”
- 1979: Hoare designs *CSP*
 - Introduces support for concurrent programming
- 1980: Alan Kay et al design *Smalltalk* at Xerox PARC:
 - First pure object-oriented language (*inheritance, polymorphism*)
 - Program statements translated to byte code, not machine code
 - Introduces virtual machine to execute byte code
 - Functions replaced by *methods*, calls by *messages to objects*



HLLs (early 80s)

- 1980: Wirth designs *Modula-2*
 - Introduces the *module* - a container for types and operations
- 1981: DOD designs *Ada*
 - Algol-family HLL
 - Introduces *generic types*, task synchronization (*rendezvous*)
- 1983: May designs *Occam*
 - Concurrent HLL based on CSP
- 1983: LLNL designs *SISAL*
 - Concurrent HLL for array-processing on supercomputers
- 1984: Sussman & Abelson design *Scheme*
 - Simplified easier-to-use Lisp, with static scoping



HLLs (mid-late 80s)

- 1984: Jordan designs *The Force*
 - First HLL for programming SIMD (vector) multiprocessors
- 1986: Stroustrup designs *C++*
 - Hybrid language: (procedural) C with object-oriented features
 - Parameterized types via macro-substitution (templates)
- 1987: Wall designs *Perl*
 - Interpreted procedural language; features of C, sh, awk, sed, Lisp
- 1988: Wirth designs *Oberon*
 - Algol-family with features for OO and systems programming
- 1988: Mehrotra & van Rosendale design *Kali*
 - First language for programming MIMD multiprocessors



HLLs (early 90s)

- 1990: Hudak & Wadler design *Haskell*
 - Hybrid language: functional + OO features
- 1990: van Rossum designs *Python*
 - Scripting language; features from Perl, Scheme, Smalltalk, Tcl
 - Focus on readability, ease for the person instead of the machine
- 1991: Microsoft designs *Visual Basic*
 - BASIC with integrated support for building GUIs
 - Later versions add OO features (classes, etc.)
 - By 2000, VB controls are most-reused “objects” in the world
- 1993: *High Performance Fortran (HPF)* released
 - Fortran extended for SIMD and MIMD multiprocessors



HLLs (mid 90s)

- 1994: *Perl-5* released
 - OO features added to Perl
- 1995: *Ada-95* released
 - OO features added to original Ada (renamed *Ada-8x*)
- 1995: Matsumoto releases *Ruby*
 - Fully OO scripting language; features from Perl, Python, ...
 - Emphasis on making menial web-development tasks easy
- 1996: Gosling et al design *Java*
 - C++ syntax, Smalltalk philosophy
 - Extensive class library (networking, graphics, threads, etc.)
 - Provides Java Virtual Machine (JVM) for platform-independence
 - Support for both applications and applets (run via www-browser)



HLLs (2000s)

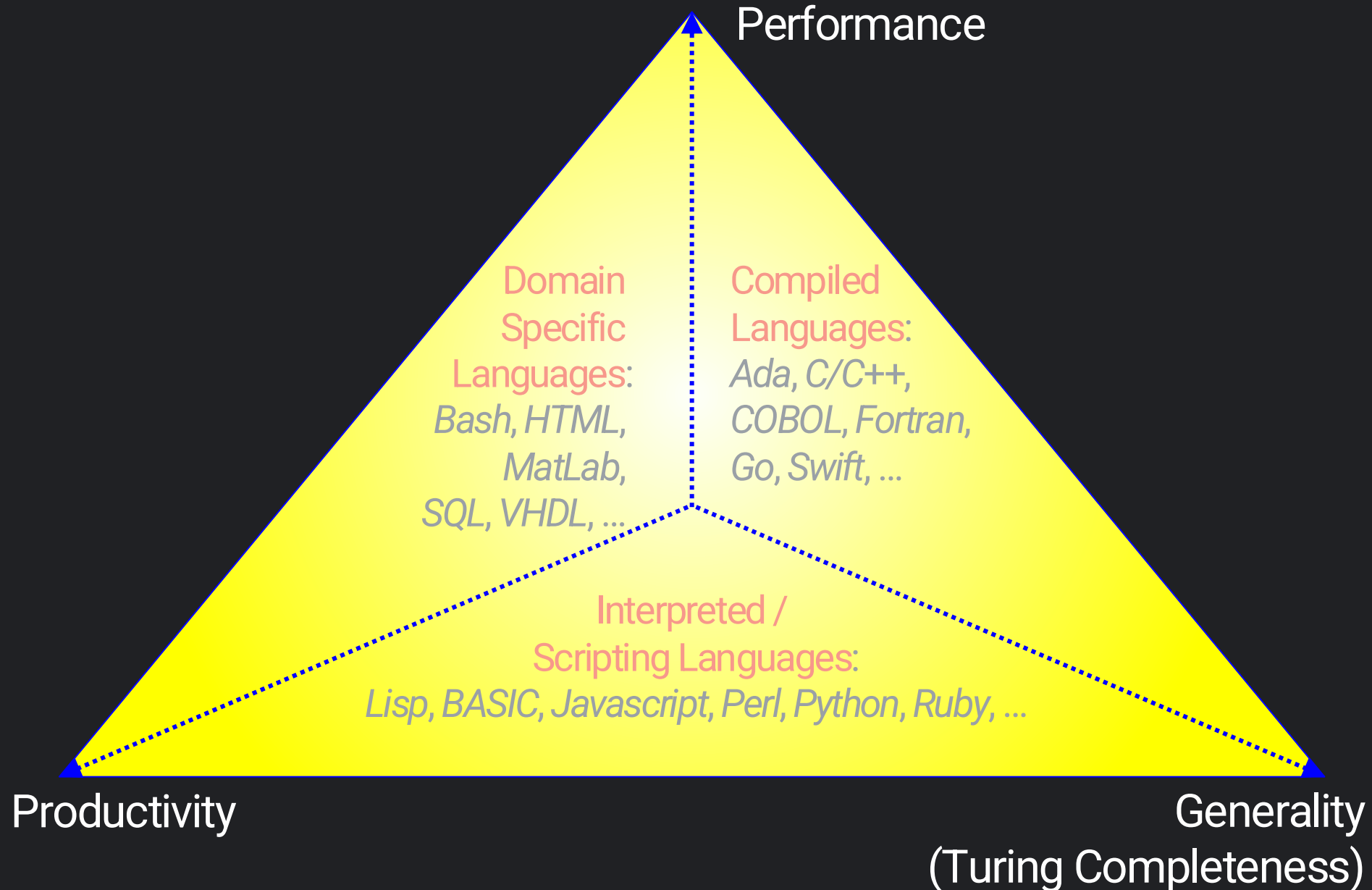
HLL development continues to this day...

- 1995: *Javascript, PHP*
- 1996: *UML*
- 2000: *C#, R*
- 2004: *Scala*
- 2007: *Go, Groovy*
- 2009: *Clojure*
- 2010: *Chapel, Rust*
- 2011: *C++11, Dart*
- 2012: *Julia, Typescript*
- 2013: *Unified Parallel C*
- 2014: *Swift*
- 2017: *C++17*
- 2018: *Fortran 2018*
- ...

... and *hundreds* of other languages along the way!



“Pick Two” Classification of HLLs





<https://craftofcoding.files.wordpress.com/2016/12/languagesptable.png>



Summary

Programming can be done in very different ways:

- **Imperative**: write blocks of *statements*
 - The Fortran- and Algol-families, Ada, BASIC, COBOL, C, ...
- **Functional**: write *functions*, pass *arguments*
 - The Lisp-family, ML, Haskell, Clojure, ..
- **Object-oriented**: build *objects*, send them *messages*
 - Smalltalk, Java, C++, Haskell, Ruby, ..
- **Concurrent**: build communicating *processes/threads/tasks*
 - CSP, Occam, Java, Ada, Erlang, Scala, Chapel, Go, Julia, ...
- **Logic**: write *inference rules*
 - Prolog, ML

These are known as the *five programming paradigms*.



Meet the languages

Java

Ada

Ruby

Clojure
(LISP)



Meet the languages: Ada

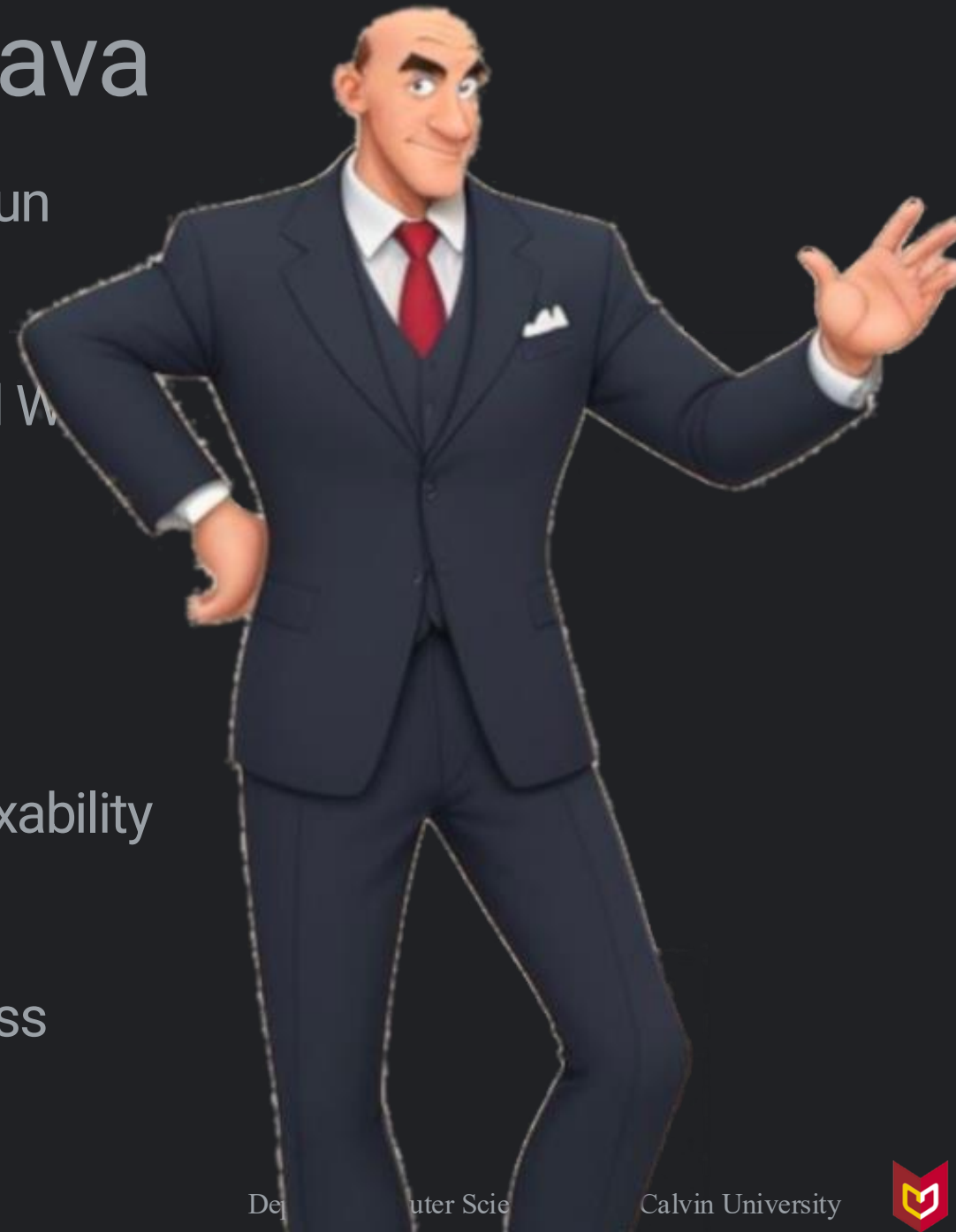


- Imperative language designed by the DoD
- Known for: Critical Systems, Military, Aerospace
- Designed for provable correctness over developer productivity
 - VERY strong typing
 - Prefers explicit detail to defaults that might be misunderstood by the developer
 - Doesn't put up with your garbage
 - (no garbage collection)
 - It will be done the ADA way, or it won't be done
- Very small compiled code
- Personality: Military Commander
- Relatives: Pascal, C++, ALGOL68
- Released: 1980



Meet the languages: Java

- Object Oriented language released by Sun Microsystems, now Oracle
- Known for: Business Applications
- Designed for developer productivity and W (Write Once, Run Everywhere)
 - Strong typing
 - Huge standard library
 - Occasionally overly formal
 - Pragmatic and effective
 - Byte code gives great balance of flexibility and performance
- Java is HUGE, Java is EVERYWHERE
- Personality: Business, Business, Business
- Relatives: C++, C#, Smalltalk
- Released: 1995



Meet the languages: Ruby



- Object oriented Scripting Language.
- Known for: Webservers, task automation
- Designed for developer productivity
 - Weak Typing
 - Many ways to accomplish a goal
 - Dynamic scripting language
 - Not fast, but not slow either
 - Friendly and flexible to the point of allowing disorganization
- Personality: Friendly, Flexible, Fun, Can-Do
- Relatives: Python, Perl, Smalltalk
- Released: 1995



Meet the languages: Clojure (LISP)

- Functional language designed to resemble mathematical notation as a language for list processing
- Clojure is Modern dialect of LISP
- Known for: AI, Research, Recursion, lists
- Easy to work with, once you get past:
 - Variables don't change value once assigned
 - Fully parenthesized prefix notation
 - EVERYTHING is a function
 - Types: list, not list
- He is recursively on his shirt, recursion is BIG
- Personality: Obsessive, hard to follow yet surprisingly effective
- Relatives: Scala, Scheme, Lua, Wolfram
- Released: 1960

