

Intro

CSCI 334
Stephen Freund

Most "Popular Languages", Jan. 2007

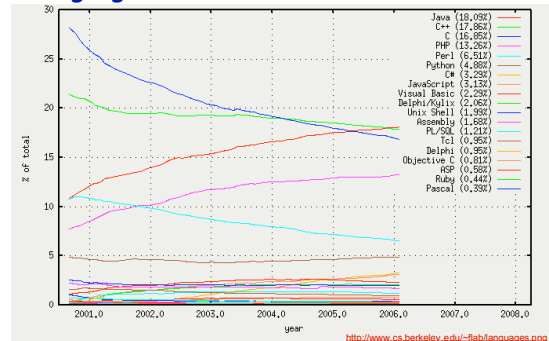
1-10	11-20	21-30	31-40	41-50
Java	SAS	Fortran	RPG	Icon
C	Delphi	Visual FoxPro	ColdFusion	Haskell
C++	PL/SQL	IDL	LabView	Natural
(Visual)Basic	D	Awk	Maple	VBScript
PHP	ABAP	Prolog	REXX	OCaml
Perl	Lisp/Scheme	dBASE	Smalltalk	Q
C#	Ada	MATLAB	Forth	Lua
Python	COBOL	Logo	CL	APL
JavaScript	Pascal	Bash	Tcl/Tk	Lingo
Ruby	Transact-SQL	ActionScript	S-lang	Objective-C

TPC index based on world-wide availability of skilled engineers, courses, and third party vendors, determined by using Google and Yahoo! search engines

and the Next 50...

- ABC, Algol, Applescript, AspectJ, Avenue, Beta, Boo, cg, Ch, Clarion, Clean, Clipper, Csh, cT, DC, Dylan, EGL, Eiffel, Erlang, Euphoria, F#, Felix, Focus, Groovy, Inform, Io, LotusScript, MAD, Magic, Mathematica, ML, Modula-2, MOO, MUMPS, Occam, Oz, PILOT, PL/1, Postscript, Powerbuilder, Progress, REALbasic, Rebol, Scala, Seed7, SIGNAL, SPSS, Verilog, VHDL, XSLT

Languages in Common Use



Position Jan 2007	Position Jan 2006	Delta in Position	Programming Language	Ratings Jan 2007	Delta Jan 2006
1	1	=	Java	19.160%	-3.10%
2	2	=	C	15.807%	-3.20%
3	3	=	C++	10.425%	-1.04%
4	5	↑	(Visual) Basic	9.123%	+0.03%
5	4	↓	PHP	7.943%	-1.46%
6	6	=	Perl	6.237%	-0.81%
7	7	=	C#	3.521%	-0.03%
8	8	=	Python	3.502%	+0.90%
9	10	↑	JavaScript	2.845%	+1.31%
10	21	11 * ↑	Ruby	2.519%	+2.15%
11	11	=	SAS	2.343%	+1.18%
12	9	↓ ↓ ↓	Delphi	2.336%	+0.75%
13	12	↓	PL/SQL	1.570%	+0.54%
14	22	8 * ↑	D	1.335%	+0.97%
15	20	5 * ↑	ABAP	1.229%	+0.82%
16	14	↓ ↓ ↓	Lisp/Scheme	0.674%	+0.07%
17	18	↑	Ada	0.638%	+0.17%
18	13	↓ ↓ ↓ ↓	COBOL	0.637%	-0.13%
19	15	↓ ↓ ↓ ↓	Pascal	0.570%	+0.04%
20	34	14 * ↑	Transact-SQL	0.510%	+0.34%

Ruby

- From Wikipedia:

Ruby is a **reflective, dynamic, object-oriented programming language**. It combines syntax inspired by **Perl** with **Smalltalk**-like object-oriented features, and also shares some features with **Python**, **Lisp**, **Dylan** and **CLU**.

- Web Services (wiki's, webmail, shopping, ...), Data Processing

Course Organization

- Programming in the Small
 - theoretical foundations
 - basic concepts
 - function, imperative
 - new ways of thinking
- Programming in the Large
 - modularity
 - program structure
 - OOP
 - concurrency, security
- Lisp, ML, Algol
- Simula, Smalltalk, C++, Java, GJ

Partial And Total Functions

- Total function $f:A \rightarrow B$ is a subset $f \subseteq A \times B$ with
 - For every $x \in A$, there is some $y \in B$ with $\langle x, y \rangle \in f$ (total)
 - If $\langle x, y \rangle \in f$ and $\langle x, z \rangle \in f$ then $y = z$ (single-valued)
- Partial function $f:A \rightarrow B$ is a subset $f \subseteq A \times B$ with
 - If $\langle x, y \rangle \in f$ and $\langle x, z \rangle \in f$ then $y = z$ (single-valued)
- Programs define partial functions for two reasons
 - partial operations (like division)
 - nontermination
$$f(x) = \text{if } x=0 \text{ then } 1 \text{ else } f(x-2)$$

Computability

- Definition

A function f is *computable* if there is a program P that computes f , i.e., for any input x , the computation $P(x)$ halts with output $f(x)$
- Terminology

Partial recursive functions
= partial functions (integers to integers) that are computable

Halting Problem

- Decide whether program halts on input
 - Given program P and input x to P ,
 - $Halt(P, x) = \begin{cases} \text{"halts"} & \text{if } P(x) \text{ halts} \\ \text{"does not halt"} & \text{otherwise} \end{cases}$
- Clarifications

Assume program P requires one string input x
Write $P(x)$ for output of P when run on input x
Program P is string input to $Halt$
- Fact: There is no program for $Halt$

Proof

- Suppose $Q(P, x)$ is a program that:
 - returns "halts" if $P(x)$ halts
 - returns "does not halt" if $P(x)$ does not halt
- Construct program
 $D(P) = \text{if } Q(P, P) = \text{"halts"} \text{ then run forever}$
 else halt
- $D(P)$ will halt if $P(P)$ runs forever.
- $D(P)$ will run forever if $P(P)$ halts.

Proof (2)

- What does $D(D)$ do?
 - If $D(D)$ halts, then $D(D)$ will run forever.
 - If $D(D)$ runs forever, then $D(D)$ halts.
- This is a contradiction!
- Therefore, our assumption that Q solves the halting problem is not valid.

Implications of Halting Problem

- There are useful program properties we cannot determine:
 - will a program run forever or not?
 - will a program eventually cause an error?
(compilers do conservative checking- more on this later)
 - will a program touch a specific piece of memory again?