Which Programming Language Is Used?

- Unix, Linux: C
- PowerPoint and Firefox: C++
- OSX, iPhone, iPad: Objective-C
- Gmail Client: JavaScript
- PeopleSoft: JavaScript+Java+SQL+
- Google Sketchup: Ruby
- Google PageRank Indexer: Sawzall
- Twitter Message Queue: Scala
- Telephone switches: Erlang
- Facebook Chat: Erlang

Most "Popular Languages", Jan. 2011

<table>
<thead>
<tr>
<th>Rank</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Java</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>4</td>
<td>C++</td>
</tr>
<tr>
<td>5</td>
<td>Python</td>
</tr>
<tr>
<td>6</td>
<td>Ada</td>
</tr>
<tr>
<td>7</td>
<td>Objective-C</td>
</tr>
<tr>
<td>8</td>
<td>Go</td>
</tr>
<tr>
<td>9</td>
<td>Lisp/Scheme</td>
</tr>
<tr>
<td>10</td>
<td>Ruby</td>
</tr>
<tr>
<td>11</td>
<td>Scala</td>
</tr>
<tr>
<td>12</td>
<td>PL/SQL</td>
</tr>
<tr>
<td>13</td>
<td>Fortran</td>
</tr>
<tr>
<td>14</td>
<td>MATLAB</td>
</tr>
<tr>
<td>15</td>
<td>Q</td>
</tr>
<tr>
<td>16</td>
<td>OOP</td>
</tr>
<tr>
<td>17</td>
<td>Smalltalk</td>
</tr>
<tr>
<td>18</td>
<td>Prolog</td>
</tr>
<tr>
<td>19</td>
<td>PowerShell</td>
</tr>
<tr>
<td>20</td>
<td>CAML</td>
</tr>
<tr>
<td>21</td>
<td>Oz</td>
</tr>
<tr>
<td>22</td>
<td>PL/I</td>
</tr>
<tr>
<td>23</td>
<td>Eiffel</td>
</tr>
<tr>
<td>24</td>
<td>COBOL</td>
</tr>
<tr>
<td>25</td>
<td>SAS</td>
</tr>
<tr>
<td>26</td>
<td>Haskell</td>
</tr>
<tr>
<td>27</td>
<td>Erlang</td>
</tr>
<tr>
<td>28</td>
<td>Progress</td>
</tr>
<tr>
<td>29</td>
<td>Alice</td>
</tr>
<tr>
<td>30</td>
<td>Awk</td>
</tr>
</tbody>
</table>

and the Next 50...

- Factor, ColdFusion, RPG, ActionScript, Groovy, FoxPro/xBase, Oberon, Natural, MUMPS,
- SIGNAL, Dylan, Occam, IDL, Maple, LabView, io, CL, NXT-G, MAD, VHDL, Limbo, Clipper,
- Ocaml, PostScript, Falcon, Transact-SQL, REXX, Lingo, Module-2, VBScript, Modula-3,
- MAX/MSP, Tcl/Tk, TeX / LaTeX, Vola, Mathematica, SuperCollider, Euphoria, AppleScript, XSLT, Genie, SISAL,
- PowerBuilder, JavaFX Script, Verilog, LabWindows/CVI, Informix/4GL, Metafont, Harbour

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

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

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 \( (x, y) \in f \) \( \text{(total)} \)
  - If \( (x, y) \in f \) and \( (x, z) \in f \) then \( y = z \) \( \text{(single-valued)} \)
- Partial function \( f : A \rightarrow B \) is a subset \( f \subseteq A \times B \) with
  - If \( (x, y) \in f \) and \( (x, z) \in f \) then \( y = z \) \( \text{(single-valued)} \)
- 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 \textit{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 \( P \),
  \[
  \text{Halt}(P, x) = \begin{cases} \text{"halts"} & \text{if } P(x) \text{ halts} \\ \text{"does not halt"} & \text{otherwise} \end{cases}
  \]
- 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 \text{Halt}
- Fact: There is no program for \text{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
  \[
  D(P) = \text{if } Q(P, P) = \text{"halts" then run forever else halt}
  \]
- \( D(P) \) 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?