Practicalities

♦ Topics.
1. Introduction and motivation.
7. The state of the art: Scala (2007)

♦ 2 exam questions.
♦ Course web page:

http://www.cl.cam.ac.uk/teaching/0708/ConceptsPL/

with lecture slides and reading material.
Introduction and motivation

References:

Why study programming languages?
- To improve the ability to develop effective algorithms.
- To improve the use of familiar languages.
- To increase the vocabulary of useful programming constructs.
- To allow a better choice of programming language.
- To make it easier to learn a new language.
- To make it easier to design a new language.

What makes a good language?
- Clarity, simplicity, and unity.
- Orthogonality.
- Naturalness for the application.
- Support of abstraction.
- Ease of program verification.
- Programming environments.
- Portability of programs.

Goals
- Critical thinking about programming languages.
- What is a programming language?
- Study programming languages.
- Be familiar with basic language concepts.
- Appreciate trade-offs in language design.
- Trace history, appreciate evolution and diversity of ideas.
- Be prepared for new programming methods, paradigms.
Cost of use.

- Cost of execution.
- Cost of program translation.
- Cost of program creation, testing, and use.
- Cost of program maintenance.

Influences

- Computer capabilities.
- Applications.
- Programming methods.
- Implementation methods.
- Theoretical studies.
- Standardisation.

Applications domains

<table>
<thead>
<tr>
<th>Era</th>
<th>Application</th>
<th>Major languages</th>
<th>Other languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960s</td>
<td>Business</td>
<td>COBOL, FORTRAN, Assembler, LISP</td>
<td>Assembler, ALGOL, BASIC, APL</td>
</tr>
<tr>
<td></td>
<td>Scientific</td>
<td>FORTRAN, Maple, Mathematica</td>
<td>JOVIAL, Forth, SNOBOL</td>
</tr>
<tr>
<td></td>
<td>System</td>
<td>BCPL, C, C++</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AI</td>
<td>LISP, Prolog</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T e X, Postscript, word processing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNIX shell, TCL, Perl</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
<tr>
<td>Today</td>
<td>Business</td>
<td>COBOL, SQL, spreadsheet, FORTRAN, C, C++, Maple, Mathematica</td>
<td>C, PL/I, 4GLs, BASIC, Pascal</td>
</tr>
<tr>
<td></td>
<td>Scientific</td>
<td>FORTRAN, Maple, Mathematica</td>
<td></td>
</tr>
<tr>
<td></td>
<td>System</td>
<td>BCPL, C, C++</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AI</td>
<td>LISP, Prolog</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T e X, Postscript, word processing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNIX shell, TCL, Perl</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell, Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
</tbody>
</table>

Motivating application in language design

A specific purpose provides focus for language designers; it helps to set criteria for making design decisions.

A specific, motivating application also helps to solve one of the hardest problems in programming language design: deciding which features to leave out.
Examples: Good languages designed with a specific purpose in mind.
- **LISP**: symbolic computation, automated reasoning
- **FP**: functional programming, algebraic laws
- **BCPL**: compiler writing
- **Simula**: simulation
- **C**: systems programming
- **ML**: theorem proving
- **Smalltalk**: Dynabook
- **Clu, ML module system**: modular programming
- **C++**: object orientation
- **Java**: Internet applications

### Program execution model
Good language design presents *abstract machine*.
- **FORTRAN**: Flat register machine; memory arranged as linear array
- **LISP**: cons cells, read-eval-print loop
- **Algol family**: stack of activation records; heap storage
- **BCPL, C**: underlying machine + abstractions
- **Simula**: Object references
- **FP, ML**: functions are basic control structure
- **Smalltalk**: objects and methods, communicating by messages
- **Java**: Java virtual machine

---

**Theoretical foundations**

Examples:
- Formal-language theory.
- Automata theory.
- Algorithmics.
- λ-calculus.
- Semantics.
- Formal verification.
- Type theory.
- Complexity theory.

---

**Standardisation**

- Proprietary standards.
- Consensus standards.
  - ANSI.
  - IEEE.
  - BSI.
  - ISO.
Language standardisation

Consider: \texttt{int i; i = (1 \&\& 2) + 3;}

\begin{enumerate}
\item Is it valid C code? If so, what’s the value of \texttt{i}?
\item How do we answer such questions?!
\end{enumerate}

\begin{enumerate}
\item Read the reference manual.
\item Try it and see!
\item Read the ANSI C Standard.
\end{enumerate}

Language-standards issues

\textbf{Timeliness.} When do we standardise a language?

\textbf{Conformance.} What does it mean for a program to adhere to a standard and for a compiler to compile a standard?

\textit{Ambiguity and freedom to optimise — Machine dependence — Undefined behaviour.}

\textbf{Obsolescence.} When does a standard age and how does it get modified?

\textit{Deprecated features.}

\textbf{Language standards
PL/1}

\begin{enumerate}
\item What does the following mean?
\end{enumerate}

\begin{enumerate}
\item \texttt{9 + 8/3}
\end{enumerate}

\begin{enumerate}
\item 11.666... ?
\item Overflow ?
\item 1.666... ?
\end{enumerate}

\textbf{DEC}(p,q) means \texttt{p} digits with \texttt{q} after the decimal point.

Type rules for \texttt{DECIMAL} in PL/1:

\begin{align*}
\texttt{DEC}(p1,q1) + \texttt{DEC}(p2,q2) & \Rightarrow \texttt{DEC}(
\texttt{MIN}(1+\texttt{MAX}(p1-q1,p2-q2)+\texttt{MAX}(q1,q2),15),\texttt{MAX}(q1,q2)) \\
\texttt{DEC}(p1,q1) \div \texttt{DEC}(p2,q2) & \Rightarrow \texttt{DEC}(15,15-((p1-q1)+q2))
\end{align*}
For \(9 + \frac{8}{3}\) we have:

\[
\text{DEC}(1,0) + \frac{\text{DEC}(1,0)}{\text{DEC}(1,0)}
\]

\[
=> \text{DEC}(1,0) + \text{DEC}(15,15-((1-0)+0))
\]

\[
=> \text{DEC}(1,0) + \text{DEC}(15,14)
\]

\[
=> \text{DEC}(\text{MIN}(1+\text{MAX}(1-0,15-14)+\text{MAX}(0,14),15),\text{MAX}(0,14))
\]

\[
=> \text{DEC}(15,14)
\]

So the calculation is as follows

\[
9 + \frac{8}{3}
\]

\[
= 9 + 2.66666666666666
\]

\[
= 11.66666666666666 - \text{OVERFLOW}
\]

\[
= 1.66666666666666 - \text{OVERFLOW disabled}
\]

---

**History**


1956–60: **FORTRAN**, **COBOL**, **LISP**, **Algol 60**.

1961–65: **APL** notation, **Algol 60** (revised), **SNOBOL**, **CPL**.


1971–75: **Pascal**, **PL/1 (Standard)**, **C**, **Scheme**, **Prolog**.

1976–80: **Smalltalk**, **Ada**, **FORTRAN 77**, **ML**.

1981–85: Smalltalk-80, **Prolog**, **Ada 83**.

1986–90: **C++**, **SML**, **Haskell**.


1996–2000: **Java**.

2000–05: **C#**, **Python**, **Ruby**, **Scala**.

---

**Language groups**

- **Multi-purpose languages**
  - Scala, **C#**, **Java**, **C++**, **C**
  - **Haskell**, **ML**, **Scheme**, **LISP**

- **Scripting languages**
  - Perl, **TCL**, **UNIX shell**

- **Special-purpose languages**
  - **SQL**
  - **\LaTeX**
Things to think about

- What makes a good language?

- The role of
  1. motivating applications,
  2. program execution,
  3. theoretical foundations in language design.

- Language standardisation.