# ∼ Topic I ∼

## Introduction and motivation

**References:**

- **Chapter 1** of *Concepts in programming languages* by J. C. Mitchell. CUP, 2003.
- **Chapter 1** of *Programming languages: Design and implementation* (3RD EDITION) by T. W. Pratt and M. V. Zelkowitz. Prentice Hall, 1999.
- **Chapter 1** of *Programming language pragmatics* (2ND EDITION) by M. L. Scott. Elsevier, 2006.

# Goals

- Critical *thinking* about programming languages.

  $\boxed{\textbf{?}}$ 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*.

# 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.
- To simulate useful features in languages that lack them.
- To make better use of language technology wherever it appears.

## 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.

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

# What makes a language successful?

- Expressive power.
- Ease of use for the novice.
- Ease of implementation.
- Open source.
- Excellent compilers.
- Economics, patronage, and inertia.

# Influences

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

# Applications domains

| Era | Application | Major languages | Other languages |
|---|---|---|---|
| 1960s | Business | COBOL | Assembler |
| | Scientific | FORTRAN | ALGOL, BASIC, APL |
| | System | Assembler | JOVIAL, Forth |
| | AI | LISP | SNOBOL |
| Today | Business | COBOL, SQL, spreadsheet | C, PL/I, 4GLs |
| | Scientific | FORTRAN, C, C++<br>Maple, Mathematica | BASIC, Pascal |
| | System | BCPL, C, C++ | Pascal, Ada, BASIC,<br>MODULA |
| | AI | LISP, Prolog | |
| | Publishing | $\TeX$, Postcript,<br>word processing | |
| | Process | UNIX shell, TCL, Perl | Marvel, Esterel |
| | New paradigms | Smalltalk, SML, Haskell, Java<br>Python, Ruby | Eifell, C#, Scala |

$\boxed{?}$ Why are there so many languages?

- Evolution.
- Special purposes.
- Personal preference.

## *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, SML Modules: 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

# Classification of programming languages

- Imperative

| | |
|---|---|
| procedural | C,Ada,**Pascal**,**Algol**,**FORTRAN**, ... |
| object oriented | **Scala**, C#,Java, **Smalltalk**, **SIMULA**, ... |
| scripting | Perl, Python, PHP, ... |

- Declarative

| | |
|---|---|
| functional | Haskell, SML, Lisp, Scheme, ... |
| logic | Prolog |
| dataflow | Id, Val |
| constraint-based | spreadsheets |
| template-based | XSLT |

# Theoretical foundations

**Examples:**

- Formal-language theory.
- Automata theory.
- Algorithmics.
- $\lambda$-calculus.
- Semantics.
- Formal verification.
- Type theory.
- Complexity theory.
- Logic.

# Standardisation

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

# Language standardisation

Consider: `int i; i = (1 && 2) + 3 ;`

**?** Is it valid C code? If so, what's the value of `i`?

**?** How do we answer such questions!?

- **!** Read the reference manual.
- **!** Try it and see!
- **!** Read the ANSI C Standard.

# Language-standards issues

**Timeliness.** When do we standardise a language?

**Conformance.** What does it mean for a program to adhere to a standard and for a compiler to compile a standard?

*Ambiguity and freedom to optimise — Machine dependence — Undefined behaviour.*

**Obsolescence.** When does a standard age and how does it get modified?

*Deprecated features.*

# Language standards
# PL/1

$\boxed{\textbf{?}}$ What does the following

```
9 + 8/3
```

mean?

– `11.666...` ?

– Overflow ?

– `1.666...` ?

`DEC(p,q)` means `p` digits with `q` after the decimal point.

Type rules for `DEC`IMAL in PL/1:

```
DEC(p1,q1) + DEC(p2,q2)
  => DEC(MIN(1+MAX(p1-q1,p2-q2)+MAX(q1,q2),15),MAX(q1,q2))
DEC(p1,q1) / DEC(p2,q2)
  => DEC(15,15-((p1-q1)+q2))
```

For `9 + 8/3` we have:

```
DEC(1,0) + DEC(1,0)/DEC(1,0)
  => DEC(1,0) + DEC(15,15-((1-0)+0))
  => DEC(1,0) + DEC(15,14)
  => DEC(MIN(1+MAX(1-0,15-14)+MAX(0,14),15),MAX(0,14))
  => DEC(15,14)
```

So the calculation is as follows

```
9 + 8/3
  = 9 + 2.66666666666666
  =    11.66666666666666 - OVERFLOW
  =     1.66666666666666 - OVERFLOW disabled
```

# History

**1951–55:** Experimental use of expression compilers.

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

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

**1966–70:** APL, SNOBOL 4, FORTRAN 66, BASIC, **SIMULA**, Algol 68, Algol-W, BCPL.

**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.

**1991–95:** Ada 95, TCL, Perl.

**1996–2000:** Java.

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

# Language groups

- Multi-purpose languages
  - Scala, C#, Java, C++, C
  - Haskell, SML, Scheme, LISP
  - Perl, Python, Ruby
- Special-purpose languages
  - UNIX shell
  - 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.