
# Programming Languages 3

2013–14

David Watt (Glasgow)
Steven Wong (Singapore)

Moodle : Computing Science → Level 3 → Programming Languages 3

University of Glasgow

# Aims

- *Syntax:* To show you how the syntax of a programming language can be formalized.
- *Concepts:* To provide a conceptual framework that will enable you to understand familiar programming languages more deeply and learn new languages more efficiently.
- *Implementation:* To explain the functions of compilers and interpreters, how they interact, how they work, and how they can be constructed using suitable tools.

University of Glasgow

# Prerequisites

- Knowledge and experience of Java
  - essential.
- Knowledge and experience of other programming languages such as Python and C
  - highly desirable.
- Understanding of elementary discrete mathematics, particularly sets and functions
  - highly desirable.

University of Glasgow

# Contents *(1)*

University of Glasgow

# Contents *(2)*

University of Glasgow

# Coursework and assessment

- Tutorial exercises (self-assessed)
- Coursework assignment (summative, **20%**)
  - extensions to a small compiler, using a compiler generation tool
- Examination (summative, **80%**)
  - syntax (10 marks)
  - concepts (20 marks)
  - implementation (30 marks)

University of Glasgow

# Programming languages

- In this course we study **programming languages** (**PLs**). — in much the same way as linguists study natural languages (NLs)
- Each PL has its own *syntax* and *semantics*. — like an NL
- PLs must be *expressive* enough to express all computations. — but much less expressive than NLs
- Computing scientists can *design*, *specify*, and *implement* new PLs. — whereas linguists are limited to studying *existing* NLs

University of Glasgow

# What is a programming language? *(1)*

- A PL must be **universal** – capable of expressing any computation.
  - A language without iteration or recursion would not be universal.
  - The *lambda calculus* – a language of recursive functions and nothing else – is universal.
- A PL should be reasonably **natural** for expressing computations in its intended area.
  - C is natural for systems programming.
  - Java is natural for applications.
  - Python is natural for scripting.

University of Glasgow

# What is a programming language? *(2)*

- A PL must be **implementable**:
  - it must be possible to run every program in that PL on a computer
  - as long as the computer has enough memory.
- A PL should be capable of reasonably **efficient** implementation.
  - Running a program should not require an unreasonable amount of time or memory.
  - What is reasonable depends on the context. E.g., Python is slow, but acceptable for scripting applications; it would not be acceptable for systems.

University of Glasgow

# Syntax and semantics

- The **syntax** of a PL is concerned with the *form* of programs: how expressions, commands, declarations, and other constructs must be arranged to make a well-formed program.
- The **semantics** of a PL is concerned with the *meaning* of well-formed programs: how a program may be expected to behave when run on a machine.
- Semantics underlies all programming, and language implementation. Syntax provides a structure on which semantics can be defined.

University of Glasgow

# Design concepts *(1)*

- **Design concepts** are the building blocks of PLs:
  - values and types
  - variables and storage
  - bindings and scope
  - procedural abstraction
  - data abstraction
  - generic abstraction
  - processes and communication *(not covered here)*.

University of Glasgow

# Design concepts *(2)*

- A **paradigm** is a style of programming, characterized by a selection of key concepts.
  - **Functional programming** focuses on values, expressions, and functions.
  - **Imperative programming** focuses on variables, commands (“statements”), and procedures.
  - **Object-oriented** (**OO**) **programming** focuses on objects, methods, and classes.
  - **Concurrent programming** focuses on processes and communication.
- Understanding of design concepts and paradigms enables us to select PLs for a project.

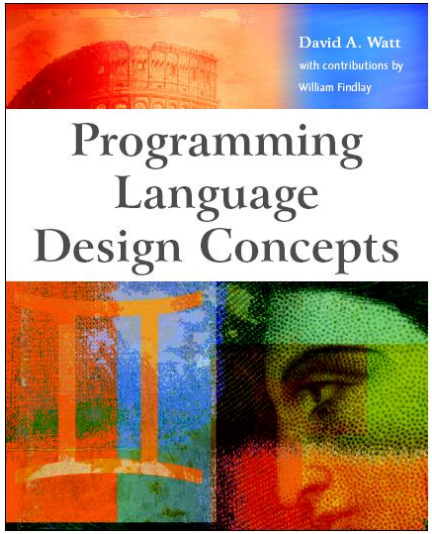
University of Glasgow

# Implementation

- A program expressed in a PL cannot be run directly by a machine. Instead it must be processed by an interpreter or compiler.
- An **interpreter** runs the given program by fetching, analysing, and executing its ‘instructions’, one at a time.
- A **compiler** translates the given program from the PL to lower-level code
- Understanding of how PLs are implemented enables us to be more skilful programmers.

# Reading *(1)*

- David Watt
  *Programming Language Design Concepts*
  Wiley 2004
  ISBN 0-470-853204
  - recommended reading for the Concepts part of this course (particularly Chapters 2–7).



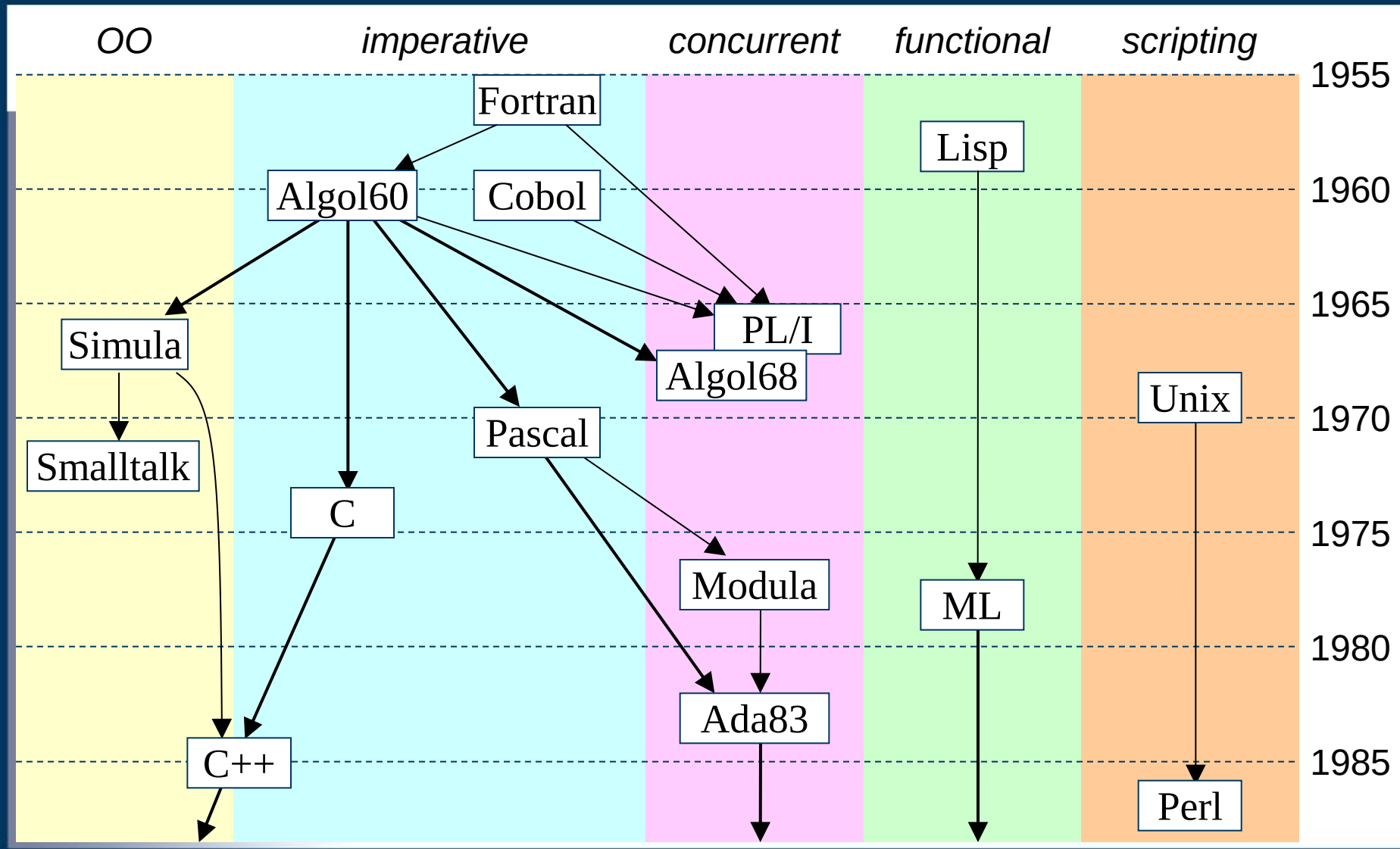
University of Glasgow

# Reading *(2)*

- David Watt and Deryck Brown
  *Programming Language Processors in Java*
  Prentice Hall 2000 ISBN 0-130-25786-9
  - background reading for the Implementation part of this course.
- Andrew Appel
  *Modern Compiler Implementation in Java*
  Cambridge 1998 ISBN 0-521-58388-8
  - additional reading
  - covers all aspects of compilation in detail, including native code generation and optimization.

University of Glasgow

# History of programming languages *(1)*



# History of programming languages *(2)*