Design and Selection of Programming Languages

SFWR ENG 3E03, Fall 2006

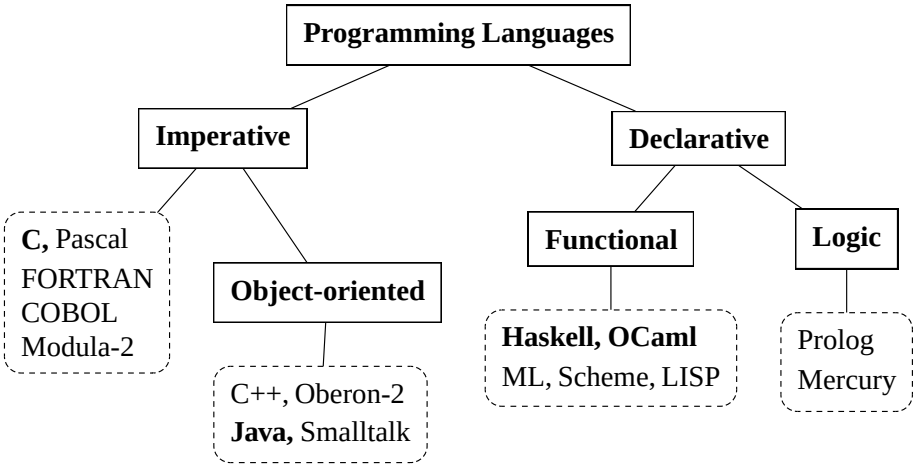
WOLFRAM KAHL

Department of Computing and Software
Faculty of Engineering
McMaster University

What Kinds of Programming Languages are There?

Imperative — "telling the machine what to **do**"

Declarative — "telling the machine what to **achieve**"



Why Study Programming Languages?

- Understand the **purpose** of programming languages
- Appreciate the advantages and disadvantages of various languages
- Understand the need for and use of **typical programming constructs**
  - this makes learning a new language much easier
- Be able to **choose** an appropriate language for a given task
- Be able to ***cope with an inappropriate language*** when forced to use one
- Appreciate that implementation aspects need to be separated from language aspects.

Are Programming Languages Important in SE?

Programming happens only in the last project phases — how can the programming language make any difference?

- Coding starts from design specifications
- **Coding considerations can influence design decisions**
- Code has to be verified against design specifications
- Code needs to be accessible to testing against specifications, requirements …
- Code needs to be accessible to **maintenance**
- Code fragments should be accessible to **reuse**

SE 3E03: Calendar Description

Fundamental structure of programming languages, language design and implementation, promote student awareness of the range of available languages and their uses.

Acquire programming skills in selected programming languages.

- Understand the fundamental structure of programming languages
- Be familiar with key issues in language design and language definition
- Be aware of the range of available languages and their uses
- Learn Haskell, learn more Java, learn more C, …

Fundamental Structure of Programming Languages

Syntax — What a program looks like

Semantics — What a program means

Pragmatics — How people use the language

Implementation — How a program executes

Historical Development of Programming Languages

Emphasis has changed:

– from making life easier for the computer

– to making it easier for the programmer.

Easier for the programmer means:

– Use languages that facilitate writing **error-free programs**

– Use languages that facilitate writing programs that are **easy to maintain**

Goal of language development:

– Developers concentrate on design (or even just specification)

– Programming is trivial or handled by computer
(***executable specification languages, rapid prototyping***)

Programming Language Paradigms

Imperative Programming Languages

Statement oriented languages

Every statement changes the machine state

Object-oriented languages

Organising the state into ***objects*** with individual state and behaviour

Message passing paradigm (instead of subprogram call)

Functional (Applicative) Programming Languages

Goal is to understand the function that produces the answer

Function composition is major operation

Programming consists of building the function that computes the answer

Rule-Based (Logical) Programming Languages

Provide rules that specify the problem solution: Prolog, BNF Parsing

Other examples: Decision procedures, Constraint(-Logic) Programming

Programming consists of specifying the attributes of the answer

SE 3E03 — Expectations

Understanding of **programming language concepts** implies the ability to **apply knowledge to new circumstances!**

- **Programming skills** in Haskell, Java, C, …
- **Reasoning skills** about programs, using formal semantics
  e.g.: "prove that *P* satisfies its specification"
- **Reading skills** in languages employing known concepts
  e.g.: "C++ has operator overloading: what does the following example do?"
- **Feature identification and comparison skills** that can be applied to previously unknown languages
  e.g.: "The reference manual of XYZ contains … compare … discuss …"

SE 3E03 — Grading

End of September?	**Midterm 1**	20% or 10%
Mid October	**Midterm 2**	20% or 10%
November	**Midterm 3**	20% or 10%
December	**Final**	(betweeen 40% and 70%)

"X% or Y%" means: If midterm result is better than the result of the final, then the midterm counts X%, otherwise Y%.

SE 3E03 — Exercises and Tutorials

- Weekly **exercise sheets**
- Some exercise questions will be similar to exam questions
- **Complete** the exercises **before** the tutorial!!!
- Tutorials are intended for ***discussion*** of ***student solutions***
- **Practice** is **essential** for acquiring ***skills!***
- Three days before an exam is **too late for acquiring skills!**

Topics

- **Syntax, Grammars, Lexing, Parsing**
- **Typing**
- **Concrete and Abstract Data Types, Classes, Interfaces**
- **Functional Programming in Haskell**
- **Programming Language Semantics**
- **Semantics-Based Reasoning About Programs**

SE 3E03 — Literature

- **Principles of Programming Languages Textbook** (optional):

  Allen B. Tucker, Robert E. Noonan. *Programming Languages: Principles and Paradigms.* McGraw-Hill, 2002. URL *http://www.mhhe.com/tucker/*. ISBN 0-07-238111-6

- **Haskell Textbook** (optional):

  Simon Thompson. *The Craft of Functional Programming.* Pearson — Addison Wesley. Second Edition, 1999. ISBN: 0-201-34275-8

- **Short Java Reference:**

  Peter Sestoft. *Java Precisely*. Second Edition. MIT Press, 2005.

- **On-Line Haskell Material:**

  http://haskell.org/, in particular “Yet Another Haskell Tutorial”