

Introduction and overview

Principles of Programming Languages

Mark Armstrong

Fall 2020

1 Preamble

The preamble section of each notes will include

- notable references,
 - i.e., specific chapters of our recommended/additional texts from which the notes are derived, or which expand on the notes,
- a table of contents, and
- an update history, chronicling any major changes.
 - Note the git commit history will provide a more fine-grained record of updates.

1.1 **TODO** Notable references

:TODO:

1.2 **TODO** Table of contents

- [Preamble](#)

2 Introduction

This section of notes introduces the course and the staff, and lays out a few central concepts.

3 TODO Welcome

4 Purpose and goals of this course

4.1 Calendar description

Design space of programming languages; abstraction and modularization concepts and mechanisms; programming in non-procedural (functional and logic) paradigms; introduction to programming language semantics.

4.2 Informal objectives

- Investigate several programming languages.
 - A relatively shallow but comprehensive survey.
 - Focusing on general-purpose languages.
- *Formally* describe programming language syntax and semantics.
 - An application of theory learned previously.
- Learn informal criteria by which to judge languages.
 - Identify what languages fit what tasks.
- Examine the origins of certain languages/groups of languages.
 - Historical context provides insight into why languages are designed the way they are.

4.3 Course preconditions

Before beginning this course:

1. Students should know and understand:
 - (a) Basic concepts about integers, sets, functions, & relations.
 - (b) Induction and recursion.
 - (c) First order logic, axiomatic theories & simple proof techniques.
 - (d) Regular expressions & context-free grammars.
 - (e) Programming in imperative language
 - (f) Basic concepts of functional programming languages.

2. Students should be able to:
 - (a) Produce proofs involving quantifiers and/or induction.
 - (b) Understand the meaning of a given axiomatic theory.
 - (c) Construct regular sets & context-free languages.
 - (d) Produce small to medium scale programs in imperative languages.
 - (e) Produce small scale programs in functional languages.

4.4 Course postconditions

After completion of this course:

1. Students should know and understand:
 - (a) The basics of several programming languages.
 - (b) Formal definitions of syntax & semantics for various simple programming languages.
 - (c) Various abstraction & modularisation techniques employed in programming languages.
2. Students should be able to:
 - (a) Reason about the design space of programming languages, in particular tradeoffs & design issues.
 - (b) Produce formal descriptions of syntax & semantics from informal descriptions, identifying ambiguities.
 - (c) Select appropriate abstraction & modularisation techniques for a given problem.
 - (d) Produce (relatively simple) programs in various languages, including languages from non-procedural paradigms.

4.5 TODO Formal rubric for the course

This was last year's rubric. It needs tweaking.

Topic	Below	Marginal	Meets	Exceeds
Familiarity with various programming languages (PLs)	Shows some competence in procedural languages, but not languages from other paradigms	Shows competence in procedural languages and limited competence in languages from other paradigms	Achieves competence with the basic usage of various languages	Achieves competence with intermediate usage of various languages
Ability to identify and make use of abstraction, modularisation constructs	Cannot consistently identify such constructs	Identifies such constructs, but does not consistently make use of them when programming	Identifies such constructs and shows some ability to make use of them when programming	Identifies such constructs and shows mastery of them when programming
Ability to comprehend and produce formal descriptions of PL syntax	Unable or rarely able to comprehend given grammars; does not identify ambiguity or precedence rules	Comprehends given grammars, but produces grammars which are ambiguous or which do not correctly specify precedence	Makes only minor errors regarding precedence or ambiguity when reading or producing grammars	Consistently fully understands given grammars and produces correct grammars.
Ability to comprehend and produce operational semantics for simple PLs	Rarely or never comprehends such semantic descriptions	Usually comprehends such semantic descriptions, but cannot consistently produce them	Comprehends such semantic descriptions and produces them with only minor errors	Comprehends such semantic descriptions and produces them without errors
Ability to comprehend denotational and axiomatic semantics for simple PLs	Rarely or never comprehends such semantic descriptions	Inconsistently comprehends such semantic descriptions	Consistently comprehends such semantic descriptions	Consistently comprehends and can produce some simple semantic descriptions

- 5 **TODO** “Principles of programming languages”
- 6 **TODO** Abstraction
- 7 **TODO** Exercises