L M U | C M S I 3 8 6

PROGRAMMING LANGUAGES

Syllabus • Fall, 2013

Tuesdays and Thursdays
10:50 a.m. to 12:05 p.m.
Pereira 200
3 semester hours

Ray Toal
rtoal@lmu.edu
Doolan 110
310.338.2773

Learning Outcomes

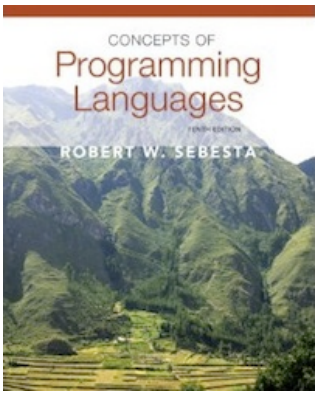
Students will

- Master some of the fundamental concepts that underlie programming language syntax and semantics through a comparative study of several languages and their features
- Learn several new programming language features and paradigms
- Gain the ability to study conceptual linguistic issues without being blinded by a particular language's implementation
- Gain insight into the problem of designing new languages

Prerequisites

Programming proficiency in one but preferably two high-level languages such as Java, C, C#, JavaScript, Ruby, Perl, or Python. Previous courses in Data Structures and Algorithms are highly recommended.

Readings



You are strongly advised to read a textbook on the field of programming languages, such as:

- Robert W Sebesta, *Concepts of Programming Languages*, 10th Edition, Addison-Wesley, 2013.

Many students benefit a great deal from language-specific books. **However, keep in mind that programming languages evolve much more quickly than books are written about them.** Here are a few suggestions for your reading pleasure; just make sure you supplement your study with online sources:

- Larry Wall, Tom Christiansen, and John Orwant, *Programming Perl*, 4th edition, O'Reilly, 2012.
- James Gosling, Bill Joy, Guy L. Steele, Jr., Gilad Bracha, and Alex Buckley, *The Java Language Specification, Java SE 7 Edition*, Addison-Wesley Professional, 2013.

- Dave Thomas, Andy Hunt, and Chad Fowler, *Programming Ruby 1.9 & 2.0*, 4th edition, O'Reilly, 2013.
- Bjarne Stroustrup, *The C++ Programming Language*, 4th Edition, Addison-Wesley Professional, 2013.
- Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, 2nd edition. Prentice Hall, 1988.
- Wesley Chun, *Core Python*, 2nd edition, Prentice Hall, 2007.
- Martin Odersky, Les Spoon, and Bill Venners, *Programming in Scala: A Comprehensive Step-by-Step Guide*, 2nd edition, Artima, 2011.
- Jeffrey Ullman, *Elements of ML Programming, ML97 Edition*, Prentice Hall, 1998.

The languages we will be covering most heavily in this course will be (1) Java, (2) C and C++, (3) Ruby or Python, (4) Scala or Haskell or Clojure or ML, (5) Rust or Go, and (6) JavaScript. Moderate attention will be focused on CoffeeScript, Ada, C#, and Perl. A smaller amount of attention will be spent on Fortran, Lisp, Scheme, SQL, Smalltalk, Lua, Groovy, D, ActionScript, and Prolog, as well as the languages from the first group of alternatives that didn't make the cut. **Note that this list includes old as well as modern languages in order to cover the evolution of ideas in the field of programming languages**.

Additional papers and readings will be assigned throughout the course (including my own course notes, practice problems, and sample code). If you have projects or papers to work on, you'll have to find some additional readings on your own. Use judgment when researching on the web; a fair amount of information is often wrong, and much of the so-called sample code is especially atrocious. Regardless, you must take the time for effective self-study.

Assignments and Grading

You'll have several homework sets containing in-depth theoretical questions and non-trivial programming problems, and quizzes and a final exam with less difficult material. To help prepare you to meet industry expectations for college graduates, programming assignments will sometimes be required to be placed in version-controlled public repositories (such as Google Code or github). To reinforce the notion of separation of content and presentation in work, you *may* be asked to create homework solutions with the LaTeX document preparation system. Exams *will* cover material from lectures *not previously assigned* for homework: don't whine about this.

Generally, coursework may be done in groups of no more than *two* students; however, while only one solution set is turned in per group, both students are responsible for understanding *all* of its content and may be asked at any time for an oral explanation of any solution. Collaboration with other groups is fine but must be limited: you may share ideas and approaches but not solutions. You **must** acknowledge any help received. Academic dishonesty may result in expulsion; be certain your work meets the standards set forth in the LMU Honor Code.

Your final grade will be weighted as follows:

Homework sets	45 pts
Quiz 1	12 pts
Quiz 2	16 pts
Final Exam	27 pts

Letter grades are figured according to the usual scale: 90% or more of the total points gets you an **A**, 80% a **B**, 70% a **C**, and so on. These are minimal requirements; for example, if you get 82 points you are *guaranteed* a **B-** or better, though you might still get an **A** since 82 may be the top score.

Homework is due at the beginning of class; late assignments are docked 30% per class. Missing class just to get an assignment done on time will not be tolerated; the only good excuses for missing class are excellent surf conditions, family problems, sickness, and personal emergencies. *Skipping class just puts your fellow students at an advantage: we often spend class time going over things that will be "on the exam".*

Where assignments involve programming, the *quality* of your code, not just its *correctness*, will play a huge part in determining your grade. I will not hesitate to assign **D**'s or **F**'s to working programs which are poorly structured, poorly commented, have poor identifier names and abbreviations, contain inappropriate hard-coded values, or are not easily maintainable for any reason. Appearance of the grading policy in this syllabus constitutes fair warning of the consequences of poorly written code.

Student Rights and Responsibilities

You have the right to:

- A syllabus with stated learning outcomes
- Have those learning outcomes addressed
- Student-instructor discussions when you need them
- Find the instructor during office hours for in-person chats
- Have questions answered via email (one-day turnaround max)
- Skype chats with the instructor, practically anytime
- Feedback on your work (and the right to challenge it or ask for more)
- A challenging experience (you should be working a little outside your comfort zone)
- Take photos of the board and record lectures
- An experience beyond what you would get from books, websites, online tutorials, and discussion forums
- Leave the course better skilled than when you came in

In return, you are expected to:

- Use the lab resources, TA time, and instructor time when you need them
- Perform work conscientiously, neatly, and in a timely manner
- Be considerate to your fellow students
- Not make lame excuses

Topics

We'll cover most of the following topics in the order listed:

- INTRODUCTION: Why study programming languages; Examples of languages with brief case studies; History and evolution; Programming paradigms; Good, bad and successful languages.

- OVERVIEW OF SELECTED LANGUAGES: Tours of [Ruby | Python], JavaScript, [Clojure | Haskell | Scala | ML], [Go | Rust], highlighting interesting and unique features; Comparisons with popular languages such as Java, C, and and C++.

- LANGUAGE SPECIFICATION: Mathematical definition of language; Syntax, semantics and pragmatics; Forms of syntax specification: CFG, BNF, EBNF, other notations; A look at semantic specification; Differences between syntax errors and static semantic errors.

- NAMES AND BINDINGS: The meaning of and importance of name, binding, scope (static and dynamic) and extent (static, stack, and heap); Application to constants, variables, types, subroutines, and modules; Shallow vs. deep binding; Closures; Aliasing, overloading, and polymorphism.

- TYPES: Type systems; Static vs. dynamic typing, strong vs. weak typing, and manifest vs. implicit typing; Type checking, type equivalence, type coercion, and type inference; Primitive types, numbers, text, enumerations, and pointers; Structs (records), unions, arrays, sets, streams, regular expressions; Abstract and generic types; Construction, assignment, equality-testing, and destruction.
- EXPRESSIONS AND STATEMENTS (CONTROL FLOW): Operator precedence, associativity, arity and fixity; Evaluation order, short-circuiting; Structured and unstructured control flow; Sequencing, selection, iteration, recursion and non-determinacy.
- SUBROUTINES: The runtime stack and activation records; Calling conventions; Passing arguments and returning values; Higher-order functions and functional programming; Closures revisited; Pattern matching for function arguments; Exceptions; Coroutines; Generic subroutines.
- ABSTRACTION, ENCAPSULATION, AND OBJECT-ORIENTATION: Modules, abstract data types; Tenets of object-orientation: encapsulation, inheritance and dynamic method binding; Issues with multiple inheritance; Implementation issues; Class-based vs. prototype-based systems; Pure vs. hybrid object systems.
- CONCURRENCY: Motivation; multiprocessing vs. multithreading; Communication and synchronization issues; Shared memory vs. message passing; Language-intrinsic concurrency vs. library managed concurrency; Implementation.

University Information

All students will want to acquaint themselves with the useful information found in the following sources:

- LMU Academic Requirements and Policies (Pay particular attention to the sections on academic dishonesty)
- LMU Disability Support Services Office
- LMU Student Codes and Policies
- LMU Learning Resource Center

Ray Toal • *Computer Science Division* • *College of Science and Engineering* • *Loyola Marymount University*